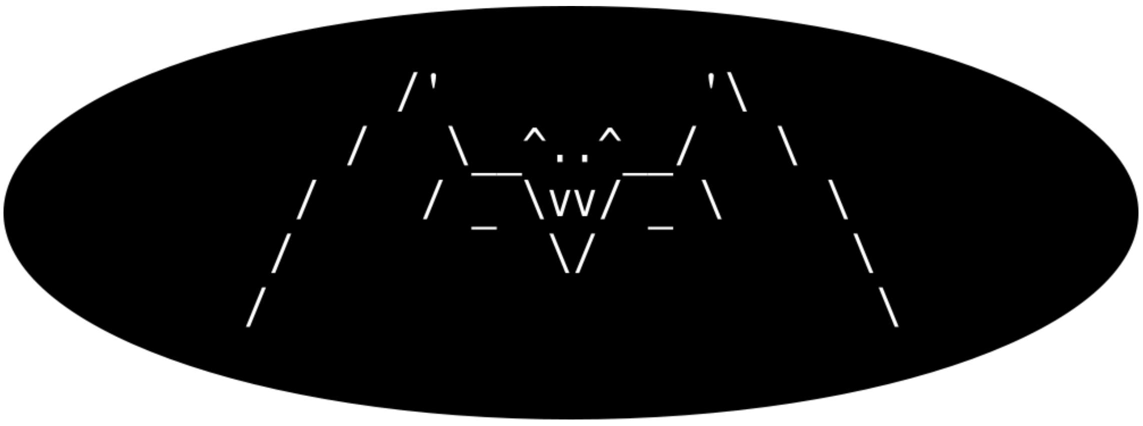
Developing a Secure Ad Hoc Network Implementation

Project Assignment



Trondheim, December 20, 2010

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics

Anne Gabrielle Bowitz and Espen Grannes Graarud

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING

PROJECT ASSIGNMENT

Students:	Anne Gabrielle Bowitz, Espen Grannes Graarud
Course:	TTM4531
Title:	Developing a Secure Ad Hoc Network Implementation

Description:

Ad hoc networks are useful in situations where wireless communication needs to be established quickly, but a secure implementation must also be able to limit access to the network. This is crucial if the implementation is to be used by e.g. emergency responders or in military applications.

In order to have a secure and restricted ad hoc network, the network must only accept communication from trusted devices. In addition the network must also know how to process unknown devices trying to gain access.

B.A.T.M.A.N. is a routing protocol for ad hoc mesh networks and we aim to extend it to support identification and authentication of mobile devices trying to access a restricted ad hoc network.

Deadline:	2010-12-20
Submission date:	2010-12-20
Department:	Department of Telematics
Supervisor:	Martin Gilje Jaatun, SINTEF ICT
Co-Supervisor:	Dr. Lawrie Brown, UNSW@ADFA SEIT

Trondheim, December 20, 2010

Stig Frode Mjølsnes, NTNU ITEM

Abstract

In emergency situations and military operations it is useful to be able to quickly establish communication. It is often necessary to accomplish this with minimum pre-existing infrastructure and without centralized administration. In such scenarios it would also be important that the network is secure - not only implying keeping the communication secret, but also be able to restrict access to the network. Wireless ad hoc networks fulfill many of these requirements, but the issue of security and access control still remains a challenging task.

The goal of this study can be divided into two parts. The first part was focused on trying to define a system with a proper authentication scheme that does not affect the nature of ad hoc networks. We combined common authentication mechanisms and an ad hoc routing protocol for this purpose. Secondly, the B.A.T.M.A.N. routing protocol was extended to incorporate the very basic functionality of the system design proposed.

A small laboratory environment was set up to test the performance of the extended protocol with the intention of proving that our basic functionality did not weaken the unique properties of mobile ad hoc networks. The test results shows that the basic idea of our system design is possible, and that the current implementation should be further extended to fulfill the requirements necessary for a secure ad hoc network.

Preface

This report is written by Anne Gabrielle Bowitz and Espen Grannes Graarud and serves as our specialization project in Information Security - a part of the 9th semester of the Master's degree Programme in Communication Technology at the Norwegian University of Science and Technology, NTNU. The assignment was proposed by Dr. Lawrie Brown of UNSW@ADFA, Australia, and Martin Gilje Jaatun of SINTEF ICT, Norway.

Writing this report has been an intriguing yet challenging task. Studying in this field of research has been very interesting and has also made us realize the significance of ad hoc networks for critical situations like emergency and military operations.

We would like to thank our supervisors, especially Martin Gilje Jaatun for the weekly meetings and feedback on our report. That was a valuable driving force and helped us greatly along the way.

Finally we would like to thank our responsible professor Stig Frode Mjølsnes from the Department of Telematics at NTNU.

Trondheim, December 20, 2010

Anne Gabrielle Bowitz

Espen Grannes Graarud

Abbreviations

AC	Attribute Certificate
AL	Authenticated List
AM	Authentication Module
B.A.T.M.A.N.	Better Approach To Mesh Ad hoc Networking
CRL	Certificate Revocation List
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
ECC	Elliptic-Curve Cryptography
EEC	End-Entity Certificate
LLPKC	Long-Lived Public-key Certificates
MANET	Mobile Ad hoc Network
MPR	Multipoint Relay
OGM	Originator Message
OLSR	Optimized Link State Routing
OSI	Open Systems Interconnection (model)
PC	Proxy Certificate
PKC	Public Key Cryptography
PKI	Public Key Infrastructure
SLC	Short Lived Certificates
SLCS	Short Lived Credential Service
UAV	Unmanned Aerial Vehicle

Definitions

Ad Hoc Network A self-organizing network with no form for pre-existing infrastructure or centralized administration.

Authenticated List A list containing the public keys, IP, roles, certificate validity period, signature fraction and the timestamp of the last received signature of all authenticated nodes in the network. The list broadcasted by the SP periodically.

Authentication Module Addition to the B.A.T.M.A.N. protocol which takes care of cryptographic functions and other additions. It also adds fields to the Originator messages which can contain a digital signature or signature fractions, and sends other messages with nonces, certificates, and ALs.

Convergence Time The time it takes for the network to get to a stable state with no route flapping after an event that has changed the network topology. E.g. a node has died or moved and made a link inferior to other alternative links.

Elliptic-Curve Cryptography Public key cryptography based on the mathematical properties of elliptic curves.

Multicast In computer networking this refers to the delivery of a packet or message to a group of devices.

Originator Synonym for a Batman interface which is a network interface utilized by Batman.

Originator Message Batman protocol message advertising the existence of an originator. They are used for link quality and path detection [NALW10].

Packet Delivery Ratio Proportion of delivered packets relative to the amount of packets sent.

Proxy Certificate A X.509 certificate signed by a regular X.509 EEC. It is used to assign roles to which the recipient can act on behalf of the signee.

Public Key Infrastructure Every entities involved with the management (creation, distribution etc.) of public key certificates. Managed by the PKIX working group of IETF.

Round Trip Delay The time it takes from a packet is sent from the sender and the sender receives as acknowledgment packet from the receiver.

Route Flapping Occurs when a node in a network continuously changes preferred route between a source and destination pair creating route instability.

Shortest Path Minimum number of hops between two communicating nodes.

X.509 Certificates Standard public key certificate standard managed by the PKIX working group of IETF.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A communications network is a critical factor in both emergency and military situations. For instance, operations like search and rescue and tactical commando operations amongst soldiers should be a coordinated effort requiring secure and quick communication among participants. The effectiveness of an operation can be heavily affected by the networks functionality, availability and reliability.

Traditional communications networks are usually dependent on some fixed infrastructure where there are physical and logical relationships between participating devices. In computer networks there are several central entities such as routers, Domain Name Servers (DNS) and Dynamic Host Configuration Protocol (DHCP) Servers that are necessary in order to have a functional network. However, in certain emergency situations or military operations, resources may be scarce, conditions unpredictable and rapidly changing, making it desirable to have a network with very little dependence to any fixed infrastructure or centralized administration.

Mobile ad hoc networks may solve many of the challenges mentioned above and some of their most important characteristics can summarized as follows:

- Self-organizing with no infrastructure or centralized administration.
- Distributed routing.
- Quick and cost-effective deployment.

Despite the advantages of using ad hoc networks, there are many challenges and issues associated with them. One in particular, which is especially important in military operations, is security. Wireless mobile ad hoc networks are more vulnerable to attacks compared to wired networks because of their highly dynamic topology, scarce bandwidth, and potentially low power and processing capacity in the devices used [MM04]. However, one of ad hoc networks biggest advantage may also be considered as the biggest challenge to a security and authentication scheme - namely the lack of infrastructure. Authentication is usually solved in a hierarchical manner on the Internet, but when a network lacks infrastructure this becomes an very difficult task.

1.1 Motivation

The motivation for this report is attempting to design an authentication scheme and access control for ad hoc networks that does not affect the networks unique characteristics. How do we balance the hierarchical structure usually required for authentication mechanisms with the flat structure of ad hoc networks? And if this issue can be solved, how can it be done without significantly reducing the performance of the network?

1.2 Objectives

The purpose of this report is first to give the reader some insight into the challenges of authentication and access control in ad hoc networks, and to give an understanding of why this is important - especially in what kind of scenarios this might be crucial.

Secondly, we aim to propose a solution for a secure and restricted ad hoc network that meets the requirements for use in emergency and military operations. The main focus will be on authentication and access control since many other security features such as key exchange for message encryption may depend on this.

Finally we will develop a very simple implementation of the authentication scheme proposed for the BATMAN routing protocol. By implementing this we aim to achieve a proof of concept that the idea of authentication can be incorporated without significantly reducing the performance or alter other functionality of the protocol.

1.3 Limitations

Our proposal is based on common security and authentication mechanisms widely used in traditional computer networks. Thus, if used right they should introduce the same level of security and trust as they do in regular networks. However, we do not known how our solution will work in a larger scale and how this might affect the performance. The system design should therefore be simulated to test the scalability and functionality under heavy load and mobility. However, due to limited time and resources we will not focus on this in this report, leaving such simulation and testing to be done in the future.

Our implementation of the proposed system design does not contain any cryptographic schemes. However, our implementation is expandable in such way that it could incorporate such functionality.

We have had limited time to look at the different security attacks on our system, so our design needs to be further evaluated in that respect.

1.4 Method

The work behind this project can roughly be split into three parts:

I. Research and study of background material.

II. Discussing and developing a proposal for a system design.

III. Modifying the BATMAN protocol to incorporate some of the basic functionalities proposed.

1.5 Document Structure

The remainder of this report is structured as follows:

Chapter 2: Background presents basic background information about ad hoc networks including network functionalities such as routing. It also sheds light on applications and challenges of ad hoc networks as well as common security issues they may face. Different authentication schemes used in traditional computer networks are also presented, such as certificates and public key cryptography.

Chapter 3: Scenarios and Requirements describes scenarios from application areas where it is beneficial and natural to utilize ad hoc networks. It also presents additional requirements that might affect how security and access control should be implemented in such networks. The chapter then continues to explain a simplified and generalized scenario based on the ones described before which is to be used as a basis for our system design, actual implementation, and testing.

Chapter 4: System Design describes in detail our proposal for a system with an authentication scheme that provides an access control mechanism for ad hoc networks. It explains how the BATMAN ad hoc routing protocol must be adapted to incorporate our authentication scheme using different authentication mechanisms described in Chapter 2.

Chapter 5: Implementation presents the BATMAN source code that we have modified and extended to implement a simplified version of the system explained in Chapter 4.

Chapter 6: Laboratory Environment and Testing explains the environment where we tested our implementation and how the testing was performed.

Chapter 7: Results presents the results from the testing done as explained in Chapter 6.

Chapter 8: Conclusion contains our evaluation of our proposed system. It highlights positive aspects in respect to the limitations of our solution. It then presents a discussion of the results received from the testing done in Chapter 6 and summarizes our findings. Finally, a short mention is done on our thoughts and ideas for future work.

The following appendices are also included:

Appendix A: BATMAN Protocol presents some additional information about the ad hoc routing protocol.

Appendix B: Lab Setup describes more thoroughly how we set up the laboratory environment.

Appendix C: Source Code presents the batman.c source code which have been extensively modified, and other code snippets from other portions of the code.

Appendix D: Test Results includes the debug output used to calculate the results from our tests.

Appendix E: Technical Essay describes the authentication process in Mobile Ad Hoc Networks and proposes a simple authentication scheme.

Chapter 2

Background

This chapter provides the necessary background required to read and understand this report. Section 2.1 describes ad hoc networks, their unique characteristics and challenges they might introduce. The chapter then continues to explain how routing is done where two different ad hoc routing protocols are used as examples. The last section 2.3 gives a short overview of different authentication mechanisms common for computer networks.

2.1 Ad Hoc Network

Definition: *"A self-organizing communications network with no form of pre-existing infrastructure or fixed centralized administration."*

A wireless ad hoc network is a collection of nodes able to communicate with each other by together creating and organizing a network. The network is able to perform regular network functions like access and routing without there being any pre-existing, fixed and centralized infrastructure. Thus nodes are not only end-entities, but must also be able to act like a router by forwarding traffic that is not destined to it self.

The nodes participating in an ad hoc network can either be fixed or mobile. Ad hoc networks where the nodes are mobile, are often referred to as Mobile Ad hoc Networks (MANETs). A closely related variant of the MANET is the mesh network where the nodes are not very mobile or not mobile at all, but they can however still join or leave the network at will making the behavior of the network similar to that of a MANETs. In this report we will focus on ad hoc networks where the participating nodes can be highly mobile, thus when the term ad hoc network or just network is used, we refer to MANET unless otherwise stated.

2.1.1 Applications

Because of ad hoc networks self-organizing nature they need very few pre-conditions in place for establishing and maintaining a network. Thus deploying an ad hoc network can be less demanding and very quick compared to other communications network like a traditional computer network [MM04]. Due to ad hoc networks' special characteristics they may find applications in several areas as briefly described below.

In emergency situations such as during war or after natural disasters, entire infrastructure-based communication may be destroyed and restoring communication quickly is crucial. By using an ad-hoc network, communication could be set up almost immediately and could then be used in emergency operations such as crowd control, search and rescue, coordinate rescue operations and so on.

Military applications also benefit from ad hoc networks ability to form a communication network quickly. In addition to this, military operations usually require a high level of security not only by encrypting the data being transmitted, but also restricting the access to the network.

The type of communication required in the environments mentioned above enforces other important requirements on ad hoc networks, such as reliability, efficiency, and support for multicast routing [MM04]. These scenarios will also be covered and further explained in chapter 3.

Other areas of use for ad hoc networks are wireless sensor networks and collaborative and distributed computing [MM04].

2.1.2 Challenges and Issues

Despite the many of advantages that ad hoc networks may introduce, several challenges and issues are also present that affect the design, deployment, and performance of such networks. Amongst the major issues relevant for our project, we find:

- Mobility of nodes
- Unreliable medium
- Resource constraints

High mobility of nodes in ad hoc networks results in frequent path breaks, packet collisions, transient loops and route flapping. This causes the network topology to change randomly and frequently making it highly dynamic. Hidden and exposed terminal problems may also appear as a consequence of the mobility of the nodes. A good routing protocol should however efficiently solve or reduce the impact of such issues [MM04].

In wireless and especially in noisy wireless areas, data packets can and will get lost. In addition, ad hoc networks with only one wireless communication interface have to cope with self-inflicted interference caused by their own wireless traffic. Thus communication links may have varying quality in terms of packet loss and will therefore also affect the network topology [NALW10].

All the issues mentioned above will in turn also affect the security in ad hoc network as explained in the next section.

2.1.3 Security Issues

Security in ad hoc networks is a challenging and comprehensive task. The unique characteristics of these networks introduce situations that are not common to traditional computer networks.

Computer networks are infrastructure-based communications networks where there exists important central entities that are necessary in order to have a functional network, e.g. routers, gateways etc. These are points in the network where it is natural to place security mechanisms. However, since ad hoc networks do not have any central entities, there are no well defined points where these mechanisms could be applied [MM04].

Nodes that participate in an ad hoc network can frequently join and leave at any point in time. If there is no authentication mechanism present, there is no association or relationship between nodes and networks making it easy for an intruder to join a network and carry out an attack [MM04].

Unlike wired networks, the communication in wireless ad hoc networks is done on a shared radio channel and may easily be picked up by any node that is in range of direct transmission. A malicious node is therefore able to perform security attacks ranging from passive eavesdropping to active message replay, and message distortion [ZH99].

Other aspects of ad hoc networks that may affect the security, is the limited resource availability found in such networks. Mobile nodes typically have limited computation capacity, battery power and scarce bandwidth which makes it difficult to implement computation-intensive tasks like complex cryptography-based security mechanisms [YLY$^+$04].

Different types of security attacks possible in ad hoc networks can be found in all the layers in the OSI model [KR09]. Some examples of attacks that can be found in the network layer are wormhole, blackhole, Byzantine, information disclosure and routing attacks. On the transport layer we can find attacks like session hijacking. In addition there are attacks which could occur in any layer, such as Denial of Service (DoS) and impersonation [MM04].

2.2 Ad Hoc Routing

As explained in [MM04] the responsibilities of a routing protocol include exchanging route information, finding a good path to a destination, discovering dead links, and restoring the paths.

Because of ad hoc networks' unique nature, classical routing protocols are typically not well suited. Thus several routing protocols have over the years been developed specifically for ad hoc networks and they can be categorized into different groups based on their properties.

In the sections below, two ad hoc routing protocols will be described and discussed. First out is OLSR which is a common and widely used protocol tailored for ad hoc networks [ZLdCG08]. The second routing protocol is BATMAN which was developed as an alternative for OLSR.

2.2.1 Optimized Link State Routing protocol

Optimized Link State Routing protocol (OLSR) is a proactive routing protocol which means that the routing in the network is done based on routing information that is periodically exchanged between nodes. The protocol belongs to the family of classical Link State Protocols which is one of two broad categories of routing protocols used in packet switching networks [CJ10].

OLSR is tailored for mobile ad hoc networks by optimizing the link state protocol in two ways [MM04]:

- reducing the size of control packets sent in the network.
- reducing the number of links that are used for forwarding link state packets.

These optimizations are realized by using only selected nodes, called Multipoint Relays (MPR), to retransmit control messages and link-state updates sent in the network. Every node in the network, called a Multipoint Relay Selector, chooses a set of neighboring nodes as their MPRs. The protocol uses hop-by-hop routing where only local information is used to route packets. This information is retrieved from the MPRs that periodically announce link-state information and control traffic to their MPR selectors. The basic layout of any packet that is transmitted in OLSR is shown in Figure 2.1.

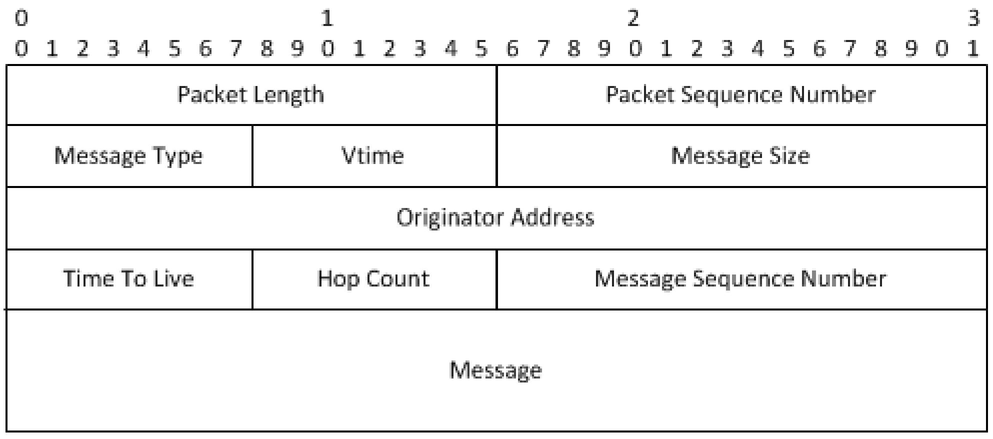


Figure 2.1: OLSR packet format omitting the IP and UDP headers.

Because of the regular transmission of control messages, the protocol is sustainable to packet loss which is common in wireless networks. Overhead of control messages in the network is also reduced and redundant control traffic is eliminated by using the MPRs [CJ10]. However, because of the many optimizations added to make the protocol more suitable for ad hoc networks, it also is more complex than e.g. the BATMAN protocol explained in the section below.

2.2.2 B.A.T.M.A.N.

B.A.T.M.A.N., or BATMAN as we will continue to write, is an abbreviation for a "Better Approach To Mobile Ad hoc Networking". It was created with the hope of being a simpler, better and more robust alternative to OLSR. The initial motivation to start the development of the protocol was mainly based on the following reasons [Mes10b]:

- The OLSR protocol seemed not to be very functional when implemented as specified in [CJ10].
- The OLSR protocol depends heavily on the assumption that every node in the network is in possession of almost the same information as all of the other nodes. As they use this information to calculate full routing path to all nodes, the more this information between the nodes differ amongst them, the more likely things like routing loops will occur.

In order to implement functional OLSR protocol to be used in real-life, the developers found themselves stripping it down removing mechanisms that were initially added to the protocol to optimize it. Eventually the developers had to break compatibility with the protocol defined in [CJ10]. A group of developers felt the OLSR protocol was becoming far to complex and decided develop a routing protocol that was simpler and better, namely BATMAN.

BATMAN is, as well as OLSR, a proactive routing protocol where every node has a routing table containing all of the nodes in the network that are accessible via single-hop or multi-hop communication links. The table, which is referred to as Originator List, does however not include the full path to a destination, only the best link-local neighbor towards it. Link-local neighbors are usually referred to as direct neighbors in the BATMAN protocol.

Nodes in the BATMAN network build their Originator Lists based on Originator Messages (OGM). These messages are small, containing only a limited amount of information such as version, Time-To-Live (TTL), sequence number, some flags and an Originator Address. The Originator Address is the IP-address of the Originator where the OGM was generated. An Originator is defined in [NALW10] as a network interface utilized by BATMAN. Every node in the network periodically generates and broadcasts OGMs for each interface it can communicate through. These messages will be re-broadcasted through the network according to BATMANs forwarding rules until they have reached all the nodes at least once. The format of an OGM is shown in Figure 2.2.

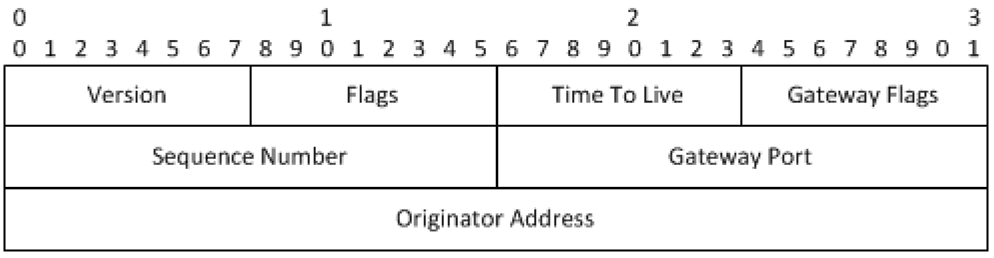


Figure 2.2: Originator Message (OGM) Format.

The best route to a certain Originator is found by counting the number of OGMs received containing this Originator Address and logging which neighbor it was received from. The best route is thus the link-local neighbor that has the highest count.

Details about the BATMAN protocol can be found in the Appendix A.

2.2.3 B.A.T.M.A.N. and OLSR Comparison

One major difference between BATMAN and OLSR is that while OLSR works to reduce the traffic load in the network by restricting which nodes that are allowed to flood, BATMAN does not care about this at all. The reasoning behind this decision is because the protocol was designed to function on unreliable media which is very unstable and can suffer from high packet loss. Thus the flooding of routing information will not saturate the network since most of the packets will be lost due to the lossy media [NALW10].

In addition, the nodes in a BATMAN network do not have to calculate the full routing path to all other nodes in the network. By only choosing the next hop towards a destination makes BATMAN a lightweight protocol that quickly adapts to the dynamic topology of ad hoc networks.

2.3 Authentication Using Certificates

In order to have a restricted ad hoc network there needs to be some form of access control mechanism in place. In traditional computer networks the issue of access control and authentication is usually solved with the use of a hierarchy of trusted third parties, and digital certificates which is associated with every entity participating in the network. A certificate contains information about the entity that defines its identity and rights in the network.

This section describes some of the different variants of the X.509 certificates that are used in X.509 Public Key Infrastructure (PKI).

2.3.1 X.509 Long-Lived Public-Key Certificates

In a Public Key Infrastructure (PKI) the conventional digital certificates are sometimes called Long-Lived Public-key Certificates (LLPKC). They are issued to end-entities by a well-known and trusted Certificate Authority (CA) that digitally signs the certificates with its private key such that it can be verified by anyone in possession of the CAs public key. Each certificate contains the public key of an end-entity and additional data such as subject's public key information, signature algorithm identifier and issuer name [Sta06].

The certificate also contains a validity period of usually months or years which is why they are referred to as "Long-Lived Certificates". This long lifetime entails that a certificate needs to be checked against a Certificate Revocation List (CRL) to ensure that it has not been made invalid whilst still in its validity period. Figure 2.3 shows an illustration of a service verifying an end-entity's LLPKC and checking it against a Certificate Revocation List.

2.3.2 X.509 Proxy Certificates

A X.509 Proxy Certificate (PC) is a conventional X.509 public-key certificate containing a critical proxy certificate information extension. The presence of this extension indicates that the certificate is a PC and that it contains one required and two optional fields; pCPathLenConstraint, proxyPolicy and Proxy Certificate Path [TET$^+$10].

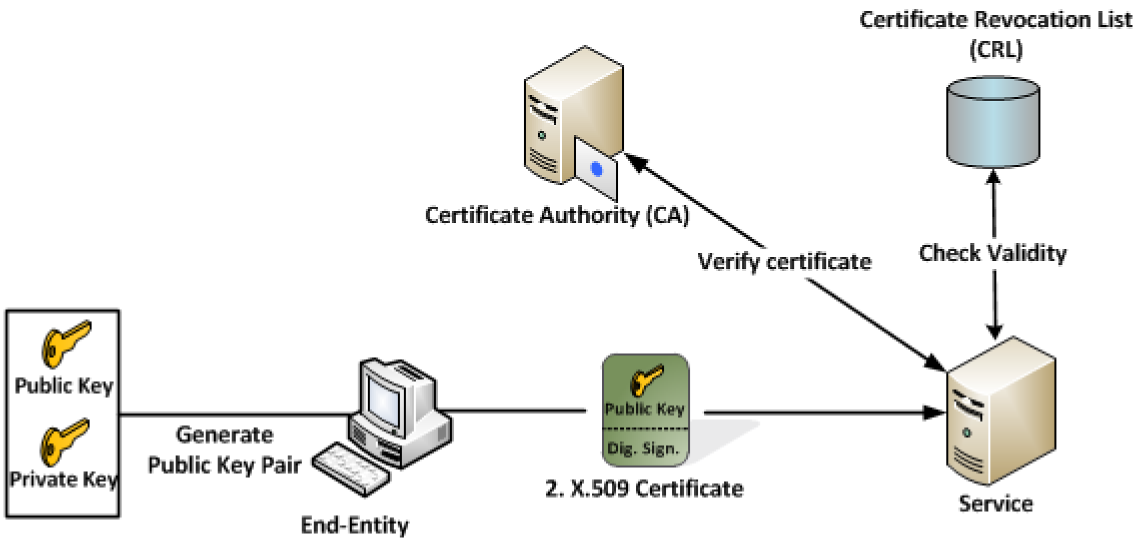


Figure 2.3: An conceptual illustration of the authentication process using LLPKCs.

The policy field in the extension can be used to make the PC a Restricted Proxy Certificate (RPC). It contains a field which specifies the appropriate language in which the policy is expressed. This option was intended to provide a finer granularity of control in the rights being delegated.

A PC inhabits some of the following properties as described in the Internet standard [TET^{+}10]:

- It is signed by an X.509 End Entity Certificate (EEC), or by another PC and is called a Proxy Issuer (PI).
- An EEC can give certain rights and restrictions to the PC it signs.
- It can sign another PC, but nothing else.
- It contains its own unique public and private key pair.
- It can be created with any desired lifetime.

An important feature of the PC is that it is given a unique identity derived from the end-entity who signed it. During the signing the PC may also inherit rights from the PI, subject to the restrictions that are placed on that PC by the PI. Thus the PC has a unique identification which can be used independently and still be associated with the PI who signed the certificate.

2.3.3 Other Certificates

Other variants of the X.509 certificate worth mentioning are Attribute Certificates (AC) and Short-Lived Certificates (SLC). ACs are certificates with a similar structure as a LLPKC, but without a PKC key pair. They contain a set of attributes tied to an identity which is used for authorization and access control decisions. ACs are usually used in association with another certificate that do contain a PKC key pair, such as a LLPKC.

The AC may have any validity period desired by the issuer and is usually has a shorter lifetime than LLPKC [FHT10].

The SLCs are modified versions of the traditional X.509 certificates and differ from these mainly because of two characteristics: certificate validity period is no more than 1 million seconds and there is no association between the client and Public Key cryptography (PKC) key pairs [HS98]. They were introduced by [HS98] with the goal of reducing the costly and difficult key management issues in typical X.509 authentication framework.

Chapter 3

Scenarios and Requirements

Before attempting to design and implement a secure and restricted ad hoc network, it is important to consider what kind of real life environments they are to be deployed in as they might introduce additional restrictions and requirements on the system.

Emergency situations and military operations are two application areas where ad hoc networks could be very useful as briefly mentioned in Section 2.1.1. In this chapter we have focused on describing an example of an emergency response scenario and a typical military scenario where we point out important restrictions and requirements. The chapter continues to describe a simplified and general scenario based on these two which is to be used during our system design and implementation.

3.1 Emergency Response Scenario

Imagine a major natural disaster has hit a densely populated area and the damages to critical communication infrastructure are high. Most of the infrastructure has been destroyed, and what is left rendered useless due to the heavy congestion put upon it.

Right after the disaster has occurred, the local Police force, Paramedics, and Firefighters begin the first phases of the rescue operation. The Police take operational control and they will then need to be able to communicate with the Firefighters and Paramedics to coordinate the operation.

Subsequently, different operational centers outside the disaster area are being setup. One center assumes the full operational role and use the local Police as a mediator to the other actors. Some centers take operational role or give assistance to their actors in the field, and some use information from the field to tell the story to the world outside.

If need be, some actors such as the Military and private actors such as The Red Cross might also show up. They will also have their own operational centers outside the area, but they are required to submit to the command centers that have the full operational role of the whole disaster area. If no such command centers are already present, they might be the ones taking over that role. Figure 3.1 depicts a full-scale scenario with on-site, as well as off-site actors. The networks in the figure are on different operational planes, shown by the vertical axis.

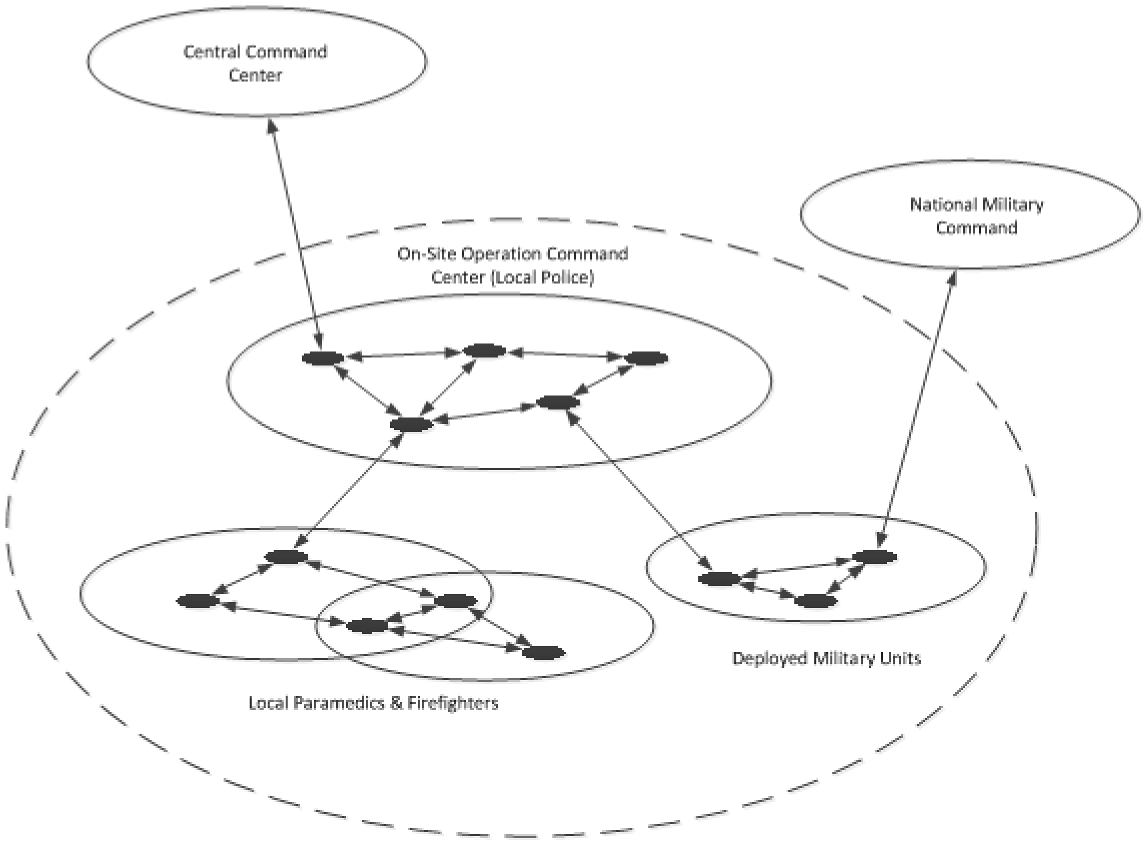

Central Command Center
National Military Command
On-Site Operation Command Center (Local Police)
Deployed Military Units
Local Paramedics & Firefighters

Figure 3.1: Disaster site with local Police running a Command Center on-site directing the other actors on the site. The figure also shows a military unit getting commands from off-site, as well as the Central Command Center which is in charge of the whole operation from outside.

3.1.1 Emergency Scenario Requirements

Much of our understanding of the emergency response example scenario is from [WSK06]. In an emergency situation there can be many different actors as shown in the example above, e.g. personnel from the Emergency Medical Services, the Fire and Rescue Services, the Police, the Military and private corporations such as the Red Cross.

In a typical civilian emergency response there will usually be Fire and Rescue Services, Emergency Medical Services and private actors in the field performing their emergency tasks. The Police will also be in the field managing the whole operation at the site. Outside the field there might be set up different coordinating centers or more professional centers coordinating specific actors in the field.

Because emergency responses are usually coordinated efforts between different actors, our system will have to make due with the fact that there might be unknown actors which are not authenticated in the usual sense, for instance if an actor is authenticated by his authority, but that authority is not yet known to the authority in charge of the operational management. If this is the case the actor should maybe have some access to the network resources because they might be very important assets to the operation. This will complicate the authorization process, and we need to determine some set of resources even unauthorized users should have access to.

In addition to local access control, we need to control the access to a gateway to the outside world, so that the coordinators outside the field can communicate with the actors in the field.

3.2 Military Scenario

The use of Unmanned Aerial Vehicles (UAV), or drones as they are often referred to in mainstream media, can help foot soldiers in the field by sending them real time video of the site. Foot soldier are thus able to see enemies that are outside of range and visible from the sky. This type of technology can give soldiers huge advantages over their opponents and must be considered very valuable.

For this to work, the UAVs need to be able to communicate between each other and with the forces on the ground. They move at a high speed, and do not (usually) operate at the same place twice. This calls for some infrastructure-less communication that can be set up on the fly, quite literally.

3.2.1 Military Scenario Requirements

A military setting will typically have higher and stricter requirements regarding the security of the ad hoc network. Here it would be natural to make sure that all nodes and users are properly authorized even to get basic network access. However, there might be some scenarios here too, where getting out information might be more important than authentication, e.g. military orders. That way, even the military might have the requirement of being able to add unknown actors to the network with limited rights.

Secure communication between nodes should be of high importance in military applications. Encrypting messages on the link layer would hide routing information and therefore implicit information of the network topology and nodes from potential adversaries. This may require us to encrypt the whole IP packets or maybe even the wireless link layer frames using 802.11i Robust Security Network (RSN) [IEE07] or WPA2 [EA04]. However, we will not go deeper into link layer encryption, as our main focus is node authentication and not confidentiality.

3.3 Our Scenario

Real life scenarios involving emergency or military operations are complex and many factors affect the situation making it almost impossible to foresee and plan for everything that might happen. The scenarios explained in the sections above presents simple situations showing important operational and organizational aspects that create important requirements that needs to be considered when designing a communications network.

Based on the two scenarios described in Section 3.1 and 3.2, we define a simpler and more general scenario focusing on the aspects that have the most impact on the communications network while also complementing all the requirements that can be identified from Section 3.1.1 and 3.2.1.

3.3.1 General Scenario Description

We imagine that nodes that participate in a communications network belong to different actors that are involved in an emergency or military situation, e.g. the police or the paramedics. To be able to identify themselves to each other they must be in possession of some authentication token, like a certificate, that proves their identity. These tokens might be given according to the different actors to which they belong or they might be issued when a node arrives to a emergency or military scene.

In order to have a restricted network there must be some nodes that are in charge of access control to the networks. It is natural to put this responsibility on the entities in the scenarios that already are in charge of operational or administration matters. These master nodes will then be the ones that verify tokens to nodes that wants to enter a restricted network and to issue new tokens to nodes that should be given access to a network.

However, it must also be possible to establish a network without the presence of these master nodes as it might not always be the case that they are in close proximity. Thus other regular nodes must be capable of taking the responsibility for the authentication of nodes in a network.

A node should be able to, in some way, join a network even though it does not have a valid token for that network. This is crucial in case the node might have something important to contribute for the e.g. search and rescue operation. However, it might be wise to give this node some limited access and rights in the network since it is not probably authenticated, but enough rights for it to be able to help.

Master nodes in different networks should be able to cooperate and merge their networks if this benefits the situation. This entails for some collaboration between these master nodes that gives their nodes access to each others networks.

In our scenario depicted in Figure 3.2 there are three nodes forming the restricted network - one node acting as a master node and one authenticated node with gateway capabilities which makes it possible to communicate with the Internet. Outside the network is an unauthenticated node that needs to be authenticated in some way to join the restricted ad hoc network. It can do so by presenting its authentication certificate to the network, and the master node will verify its identity and decide whether to grant it access or not.

3.3.2 Requirements

Our scenario generates some simple requirements, which we will have to address in our system design. The requirements are summarized in the following table:

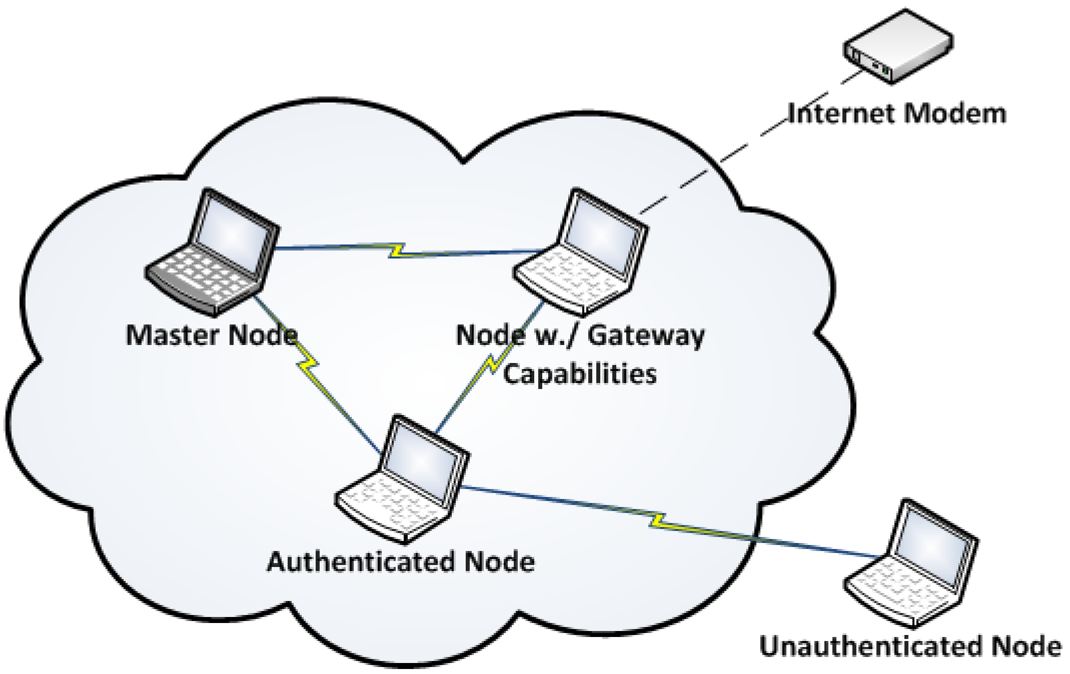


Figure 3.2: Small restricted network with one master node, one authenticated node with gateway capabilities, and one unauthenticated node.

Requirement	Requirement Description
R1	A node must be properly authenticated to get full rights in a network
R2	A node which is not properly authenticated should only get limited rights in the network
R3	A node which is not properly authenticated should be able to full rights after receiving the correct network token
R4	All networks should have a master node which handles access control
R5	All nodes should be able to assume the role as master node or limited master node
R6	Different networks should be able to collaborate

Table 3.1: Requirements based upon our simplified and general scenario.

Chapter 4

System Design

This chapter presents our proposal for how a secure and restricted ad hoc network could be accomplished. It will cover which mechanisms described in Section 4.2 we combine to ensure a secure and restricted ad hoc network and how they must be tailored to handle the special issues these kinds of networks introduce. The chapter then goes into details about how challenges regarding Public Key Infrastructures (PKI) and Certificate Authority (CA) hierarchies can be circumvented. Finally it gives the reader a technical description of how the BATMAN routing protocol should be modified and extended to handle the authentication.

4.1 Design Overview

Entities included in our system are much based on the ones described in Chapter 2 and here we briefly describe the main roles of the entities and terminology used in our solution:

- **Service Proxy (SP)** is responsible for tasks similar to that of a CA or a node with an end entity certificate (EEC) as explained in Section 4.2. The SP is in possession of a Long-Lived Public Key Certificates (LLPKC) and has the ability to verify other LLPKCs belonging to nodes entering the network that the SP is managing. It is also allowed to sign Proxy Certificates which will be issued to nodes after they have been through an authentication process with the SP.

- **Proxy Certificate (PC)** indicates a proxy certificate as described in Section 2.3.2 and more thoroughly in [TET$^+$10]. In our system the term PC indicates a proxy certificate generated by a node that has not been signed yet. Depending on which entity that ends up signing the certificate, it will be named PC0, PC1 or PC2 as explained below.

- **Proxy Certificate 0 (PC0)** is a proxy certificate belonging to a SP and is self-signed by the SPs LLPKC. This PC will have the certificate depth of 0 thus we refer to it as a PC0.

- **Proxy Certificate 1 (PC1)** is a PC that can only be signed by a SP. The certificate is only signed if the node has as valid LLPKC. A node in possession of a PC1 is fully trusted node in the network managed by the SP who signed the certificate. It is not allowed to verify other LLPKC, but is delegated the right to sign PC2s which is explained next.

- **Proxy Certificate 2 (PC2)** works in the same way as PC1 but with limited rights in the network. The restrictions put on the certificate are explained later in 4.5. The PC2 is either signed by a node in possession of a PC1 or the SP itself.
- **Authentication List (AL)** is a list containing the necessary information about all the authorized nodes in a network. All nodes have a local AL which they use to decide whether they will process Originator Messages (OGM) received by neighbors. Their local lists are updated by an AL which is periodically broadcasted to all the nodes in the network by the SP. More details about this list is explained in Section 4.4.

In our system, the OGMs sent by the BATMAN protocol have been modified to accommodate for additional information which will be used for authentication in the network. The authentication information is appended to an OGM by an Authentication Module (AM). For further referencing, we divide OGMs sent by a node in two types:

- **Plain OGM** refers to a regular OGM that does not contain any additional information added by the AM. This indicates that the OGM is sent from a node that does not belong to any network yet.
- **OGM** indicates that the OGM was broadcasted by a node that has been authenticated somewhere and is thus part of some network.

More about the format of the modified OGM and the AM is explained in Section 4.3.

The essential idea of the system design explained in this chapter is that the nodes participating in a restricted network will only process routing updates from authorized nodes which have valid proxy certificates and are listed in their local Authentication List (AL). The example described in the next section illustrates the basic functionality of our system.

The following sections explain more thoroughly the details about the how the authentication is done, gives a technical description of the modified BATMAN protocol and certificates used, and finally a discussion of the solutions challenges and limitations.

4.1.1 Simple Example

This example can be divided into three parts that show different important aspects of our solution.

Part I Let us consider a very simple example where we have one SP present and one unauthenticated node A, both with empty ALs. The SP and node A will periodically broadcast OGMs as usual as part of the BATMAN protocol. Node A will broadcast plain OGMs as it is not part of any network yet. Upon reception of the plain OGMs from node A, the SP will engage in a handshake with A where node A is eventually issued a PC2 that is signed by the SP.

This is the general course of events for every node that enters a network it is not already authenticated in and is within transmitting range of SP. Figure 4.1 is a message sequence chart (MSC) illustrating the messages exchanged in this scenario.

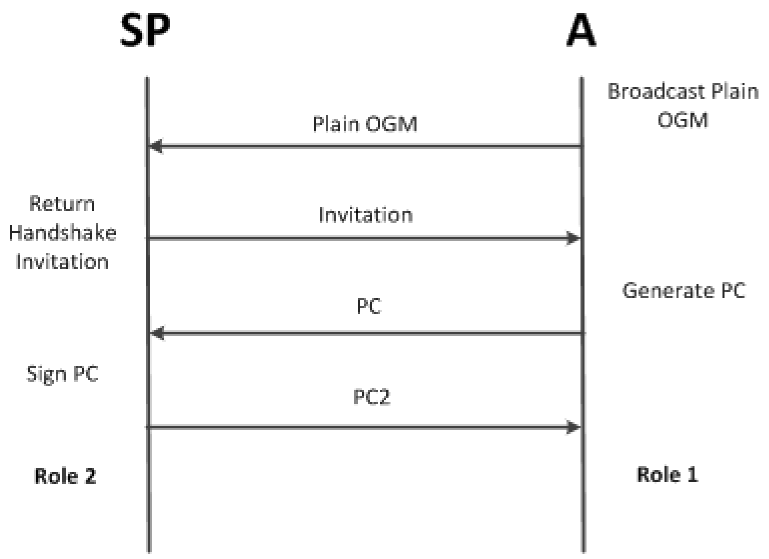


Figure 4.1: Initial handshake between the SP and Node A, which results in Node A getting a PC2.

Part II After node A has received its new certificate, it will now start using the additional authentication fields in its OGMs to show that it is in possession of an PC2.

When the SP receives the new OGMs from node A, it knows that A might be able to be upgraded with a PC1. So upon the reception of an OGM from A, the SP will again invite to a new handshake which is somewhat similar to the one described above, only here the nodes also validate each others LLPKC. The authentication process is finalized when node A is issued a PC1 signed by the SP and becomes a fully trusted node in the network.

Figure 4.2 shows how the network is established and how the nodes ALs and certificates change during the different steps in the authentication process.

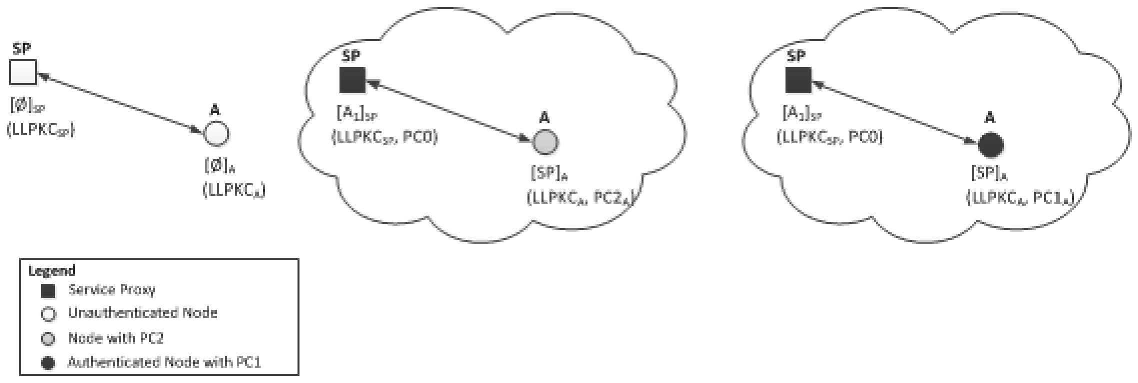


Figure 4.2: Illustrates stepwise how a network is established between a node A and SP, both with valid LLPKC

If a node possessing a PC2 and a LLPKC is a direct neighbor with the SP, the following course of events will be as depicted in Figure 4.2. This is of course only true if the SP can recognize and verify the PC2 and the LLPKC.

Part III If however an unauthenticated node is not within transmitting range of the SP, the plain OGMs might still reach some of the other nodes in the network. To be able to authenticate this new node, we use the fact that nodes in possession of a PC1 are allowed to sign PC2 certificates. To show how a new node can join a network without being in direct contact with the SP, let us imagine the following scenario: Node B now wants to join the restricted network established by the SP and node A shown in Figure 4.3 and it is only able reach node A with its OGMs.

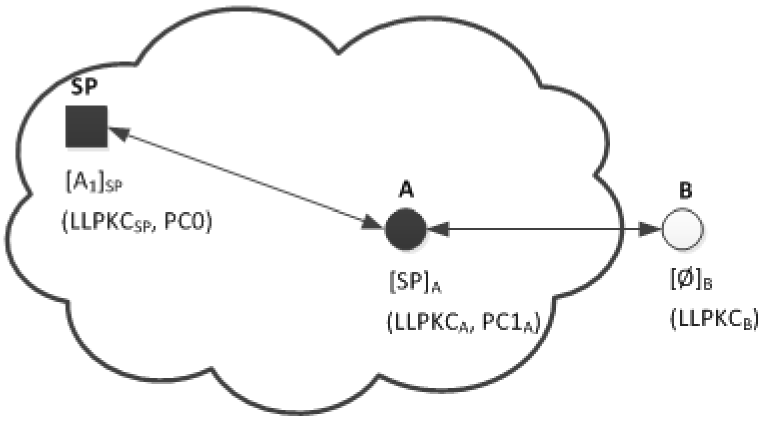


Figure 4.3: The figure shows a new unauthenticated node B entering a restricted network with node A an SP. Also depicted is a simplified AL and certificates.

Because node A has a PC1 it is able to sign and issue a PC2 to B such that it is now untrusted member of the network. The OGMs sent from node B will eventually reach the SP through A and the SP is also able to verify the OGMs because it has a PC2 signed by As PC1. The SP will then recognize that since node B only has PC2 it may be qualified to be upgraded with a PC1. Thus the SP will initiate a handshake similar to the one explained in Part II. If it has a valid LLPKC, node B will be issued a PC1 and also become a fully trusted member of the network.

4.2 Authentication

This section presents in detail how the authentication process is done when exchanging and verifying certificates between nodes. It also goes into detail on what is actually sent in the messages during the two different handshakes mentioned in the previous section, which have been called Initial and Authentication Handshake respectively.

4.2.1 Initial Handshake

The first handshake that an unauthenticated node is involved in is between the new node and the SP as depicted in Part I in the example above. This handshake is called an initial handshake.

During this initial handshake, the unauthenticated node will generate a public key pair and a corresponding PC. It sends this PC to the authenticated node which will in turn sign it with its private key from PC1. The PC then returned is thus what we call a PC2. The process is the same if the signing node is the SP, the only difference is that the private key from its PC0 is used to sign the certificate, and the new node will still get a PC2. When this step is complete the new node will be able to use his PC2 to identify itself as a node with limited rights in the network, and also use it for encryption in a new handshake.

Message Exchange in Initial Handshake

Here we show more detailed what information the Authentication Module (AM) exchanges during a initial handshake between a new unauthenticated node A and a regular authenticated node B. Public and private keys are denoted as PU and PR respectively, and they belong to the certificate being used by the sender node. A nonce is represented by N, encryption E and Proxy certificates are denoted PC. The handshake is triggered by B receiving a plain OGM from A and continues as follows:

I $B \rightarrow A : N_B$

II $A \rightarrow B : PC_A\{PU_A\} \,||\, E_{PR_A}(N_B)$

III $B \rightarrow A : PC2_A\{PU_A,\, E_{PR_B}(Hash[PC_A])\}$

The nonce value sent in message I is the invitation to the initial handshake. Node A responds with a self-generated PC and proves it owns the certificate by using the corresponding private key to encrypt the nonce value. Node B checks the nonce value by decrypting the message and comparing it to the sent nonce with the public key extracted from the PC, $D_{PU_A}(E_{PR_A}(N_B)) = N_B$. Then node B signs the certificate, effectively making it a PC2 and sends it back to node A. It is important to notice that B never reveals its PC1 or public key.

What is shown in this list is a minimum requirement of what should be put in the messages sent. Certificates such as the PC should however contain much more information than just e.g. the public key. However, it is not necessary to include this information here since it has no impact on how the handshake is performed, but it will be discussed in Section 4.5.

4.2.2 Authentication Handshake

If the SP discovers a new node with a PC2 in its network, it will invite the node to what we called an authentication handshake. This handshake is shown in both Part II and III in the simple example above. The goal of the handshake is to establish full trust between the nodes with a mutual authentication where they verify each others' LLPKC. In this handshake, the SP will make use of the encryption channel established in the initial handshake to protect and reduce the exposure of important information, such as the LLPKC.

Message Exchange in Authentication Handshake

The message exchange in this section is denoted in the same manner as previously. The only difference is that we also handle LLPKC certificates.

The first three messages shown below will always take place during the authentication handshake. The important thing that happens here is that node A sends its LLPKC to the SP. The authentication handshake is initiated when the SP receives an OGM that indicates that it has a PC2 and no PC1. This is shown in message I where a digital signature of the OGM is appended. This we will come back to later.

I $A \rightarrow SP : OGM_A \,||\, E_{PR_{PC2_A}}(Hash[OGM_A])$

II $SP \rightarrow A : E_{PU_{PC2_A}}(N_{SP})$

III $A \rightarrow SP : E_{PR_{PC2_A}}(LLPKC_A,\, E_{PR_{LLPKC_A}}(N_{SP},\, N_A))$

The next step of the handshake depends on whether the SP finds the LLPKC from node A valid. If it does so, the SP will return its own LLPKC back to A for mutual authentication. The observant reader will notice that the public keys for the LLPKC and PC0 of SP is never revealed to an untrusted node.

IV $SP \rightarrow A : E_{PU_{LLPKC_A}}(LLPKC_{SP},\, E_{PR_{LLPKC_{SP}}}(N_A,\, N_{SP}))$

If node A can verify the LLPKC of SP, the handshake is continued. Otherwise it will be aborted at this point. The rest of the handshake is similar to the initial handshake, except that it will be done in privacy with help of the keys from the LLPKCs.

The last messages below show that A generates a new PC which is signed by the SP making it a PC1. The last message of the handshake provides node A with its own PC1 and the PC0 of the SP such that it can verify routing messages from the SP.

V $A \rightarrow SP : E_{PU_{LLPKC_{SP}}}(PC_A\{PU_{PC_A}\})$

VI $SP \rightarrow A : E_{PU_{LLPKC_A}}(E_{PR_{LLPKC_{SP}}}(PC1_A\{PU_{PC1_A}, E_{PR_{PC0_{SP}}}(Hash[PC_A])\}, PC0_{SP}))$

4.3 Authentication Module

This section show how the original BATMAN protocol should be modified to incorporate our authentication scheme. Here we introduce the Authentication Module (AM), explain how it changes the OGMs and the general BATMAN routing flow. The AM is responsible for handling three things: OGM signatures, handshake messages, and broadcasting of Authentication Lists which will be described later.

4.3.1 OGM Signature

The Originator Message (OGM) as described in the original BATMAN protocol, has been modified such that it includes the number and lengths of the fields appended by the Authentication Module. These fields are used to contain signatures that a node uses to authenticate itself when sending or rebroadcasting an OGM. A node should be able to append several signatures as it can be a member of several networks using different PCs with different key pairs, e.g. PC1 in one network and PC2 in a different one.

The signature is a encrypted hash made from all static fields in the OGM, i.e. version number, sequence number and the originator address, in addition to the sequence number.

Hashing and signing an OGM every time it is sent between nodes would introduce a high computational cost and be time consuming for the resource limited nodes. Additionally, it would add a lot of network load, and it is difficult to predict how well the network would handle all the extra routing traffic load. A suggestion would then be that the signature should only be generated periodically. I.e. if we chose a time interval of say 60-120 seconds, the signature will only changed and sent for every 60-120 seconds that has passed. In between the time interval, only a small number of bits constituting the most significant bits of the signature will be appended to the OGM.

When a node receives an OGM from its neighbor for the first time, or when the node has produced a new signature, it checks the signature by calculating the hash of the OGM and comparing it to the decrypted signature appended to the OGM. If they are the same, the fraction of most significant bits is stored alongside the neighbor node in the local AL.

For the next OGM it receives, it will check its local AL for the most significant bits of the signed hash that belongs to this node and compare it to the signature fraction appended to the OGM. If the signature fractions are equal, the node is still authenticated and the OGM will be processed. If not verified, the OGM will be dropped. The hash values stored in the AL must be updated ever time a node generates a new signature for its OGMs.

Because of the likelihood of packet loss in ad hoc networks, the receiving neighbor node does not require to see a new signature from the originator at the same rate as the originator is sending them. It only needs to see a new signature (and new subsequent fractions) within a reasonable window period, e.g. 3-5 times the signature update frequency. However, if that window passes and no new signature is received, it will start dropping the OGMs and remove the node as a direct neighbor.

The interval for when a node should update its signature and send OGMs containing signatures, is a trade-off between security in the likelihood of replay attacks i.e. how many replayed OGMs is needed to change the routing topology significantly, and of computational cost.

4.3.2 Handshake Messages

The Authentication Module has the responsibility of the handshakes as explained in Section 4.2. I.e. it reacts on plain OGMs, creates nonces, has the cryptographic functions and so on. When the module starts or reacts to a handshake, it does so in parallel of normal of the BATMAN operations. Handshake messages are sent using plain UDP datagrams, completely separated from OGMs. The UDP datagrams are also directly addressed to the recipient is possible, and not broadcasted to every node in the network.

4.3.3 Authentication List Broadcast

As with handshakes, the authentication lists are completely handled by the Authentication Module. The AM maintains the local copy and takes care of broadcasting the AL in UDP datagrams.

4.4 Authentication List

The authentication list is used for continuous node authentication during routing. The SP is responsible for broadcasting this list periodically such that every node in the network will have an updated local list over nodes that have been authenticated in the network. The AL is a table where every node is represented with a row which should at least contain the following elements:

- **Node ID** The unique idenity from the proxy certificate which is an unique identifier of the node owning the certificate. The ID ties the correct node to the right signature.
- **Originator address** IP address of the authenticated node.
- **Public key** Public key of the corresponding node.
- **Role** Indicates whether the originator has a PC0, PC1 or PC2 certificate. This will affect how an OGM is processed after its sender has been authenticated.
- **Validity Period** The lifetime of the nodes PC.
- **Digital Signature Fraction** The fraction of the most significant bits of the last verified signature.
- **Last Signature Received** Timestamp of the last received signature.

Upon broadcasting the AL, the SP will encrypt the list with its private key from its LLPKC guaranteeing its confidentiality and integrity as well as signing it afterwards with its PC0 for authenticity. Every node in the network has an explicit or implicit trust of the SP to sign and verify new nodes; therefore every node will trust the content of this list even if it is received through another node in the network.

We differ between two types of Authentication Lists - one which is a complete list of all the authenticated nodes in the network, and one which is only a single entry update containing new nodes that have been issued a PC2 from a trusted node, or PC1 from a SP.

When a trusted node has signed a new nodes PC2, it needs to make the other nodes aware of this node thus needs to be able to update the other nodes AL with this node. The trusted node then makes use of the single row AL, and signs the AL update with its PC1. It does not however, encrypt the message as the SP would do with the full list. Encryption of this update is not necessary, as the information about a new node with PC2 is not regarded as confidential. Neighboring nodes will add this new node to their local ALs and rebroadcast the list until every node in the network has it. When every node has received the AL update, they will be able to forward the new nodes OGMs, which will then finally reach the SP and trigger the authentication handshake.

A node with PC2 is not able to read the AL that is broadcasted from the SP as it is not in possession of the public key of SPs LLPKC that can decrypt the list. This way, we still protect the public keys of the SPs and trusted nodes.

4.5 Certificates

The certificates used in our system design are Long-Lived Public Key Certificates (LLPKC) and different types of Proxy Certificates (PC). The reasoning for why we have chosen to use these certificates is explained in this section.

4.5.1 Long-Lived Public Key Certificates

These certificates are used for identity authentication which may, if recognized, give the owner full access rights to the network. It is also in these certificates stated explicitly whether the owner can assume the SP role. These certificates will typically be issued before arriving at the scene of the BATMAN network. The LLPKC may be regular X.509 certificates as explained in Section 2.3.1.

4.5.2 Proxy Certificates

One of the benefits of choosing proxy certificates in our system is their ability to define nodes rights in the network in their assigned certificates. By utilizing the Restricted Proxy Certificate (RPC) explained in Section 2.3.2 these rights are decided by the node who signs the certificate and are be specified in an appropriate language. A PC can be created with any desired lifetime and will in our case typically be given a lifetime that is significantly shorter than the LLPKC. This fits well with the highly mobile nodes that are assumed to be in these kinds of networks and the environments in which they are to be used.

As mentioned in the design overview in section 4.1, we differ between three types of proxy certificates: PC0, PC1, and PC2. PC0 is the certificate with the highest rights and is self-signed by the SPs own LLPKC. It is able to establish a proper network and to verify and sign new PC1s to authorized and trusted nodes. In addition the node in possession of a PC0 is also allowed to broadcast the full Authentication List.

PC1 on the other hand is not allowed to sign new PC1s, but can sign PC2. This is such that node can be able to join the network without being in direct contact with the SP. A PC1 is also allowed to broadcast a AL containing a new PC2 that it has just allowed into the network. This must be possible because the OGM then sent from the node with PC2 must be allowed to be forwarded in the network as well such that it will eventually reach the SP. This way, it can be authenticated properly if it has a valid LLPKC, and the network is also able to grow without being too dependent on the SP. It also relaxes some of the load on the SP.

A node with a PC2 has the most restricted role in a network. Trusted nodes in the network will only accept the PC2s own OGMs, not if it rebroadcasts any others OGMs. Thus the network will be aware of the node with the PC2, but not any other networks the node might be connected to.

As everyone will be aware of the node, the SP is able to do a new authentication handshake and check if it can be upgraded with a PC1. But until it does, it will have limited influence on the existing network and the routing done within it.

A specific cryptographic scheme for the proxy certificates has not been decided yet, but Elliptic Curve Cryptography looks promising given its short keys. As the SPs have to periodically broadcast the public keys in the ALs, keeping the size to a minimum is of importance.

4.6 Service Proxy Presence

This section explains what happens if there are no service proxy (SP) present when establishing a network. It also describes what happens when a SP enters a network which is established by only regular nodes.

Two nodes discover each other in such a situation by broadcasting Plain OGMs as normal. They may both be in possession of LLPKCs, but there is however no SP present that can verify them. Even though there is no SP in the network, it is preferred that one of the nodes should become a master node which would have similar but limited rights as the SP.

How this role is assigned can be done in several ways. One suggestion is mentioned in an essay that can be found in Appendix E. However, to keep things simple in our implementation, an easy way to implement this would be to let the node with the lowest link layer address assume the master role.

The master node could generate a self-signed certificate, PC0 for which it can use to sign new PC1s to nodes entering the network. It is up to the master nodes whether it will allow every node that it discovers access to the network, or if they must have LLPKCs signed by the same CA as itself. We leave this choice to the users of the protocol because this depends on the setting in which the protocol is used.

After such a network has been established, a SP may join at a later point in time. When the SP is in close proximity of the network, it will immediately start receiving the OGMs that are broadcasted in the network. As it is not able to recognize and verify the OGMs sent, it will automatically initiate a handshake with these nodes, issuing PC1 or PC2 depending on what kind of certificates they have. This way the SP will effectively be able to take over the management of the network after some convergence time.

It is important to consider such scenarios as this, because it might not always be the case that a node that can act as a SP is present at a location. It is then important that the nodes are still able to configure a network between them

4.7 Network Merging

When two networks are within the same vicinity they may want to be able to merge such that they are able to co-operate or to use each other as a resource. Given the scenarios described in Chapter 3 the reason for merging becomes even clearer. This section will discuss how a limited merging will happen automatically as a consequence of how the protocol/system works now and how it can be extended to make a more complete merging.

4.7.1 Limited Merging

A node which is a trusted member in one network, i.e. has a PC1 here, is also able at the same time to have a PC2 belonging to another network.

The node will continue to broadcast its OGMs as usual, but now appended with signatures made with both proxy certificates. However, it will only rebroadcast OGMs from the network where it is fully trusted and not in the one where it only has a PC2, as defined in Section 4.5. The node will also only use the PC1 connected to the specific network in which it sends the rebroadcast. I.e. it will only add the PC1 from the first network in its rebroadcasts of OGMs from that network, and it will discard OGMs from the new network where it only has a PC2.

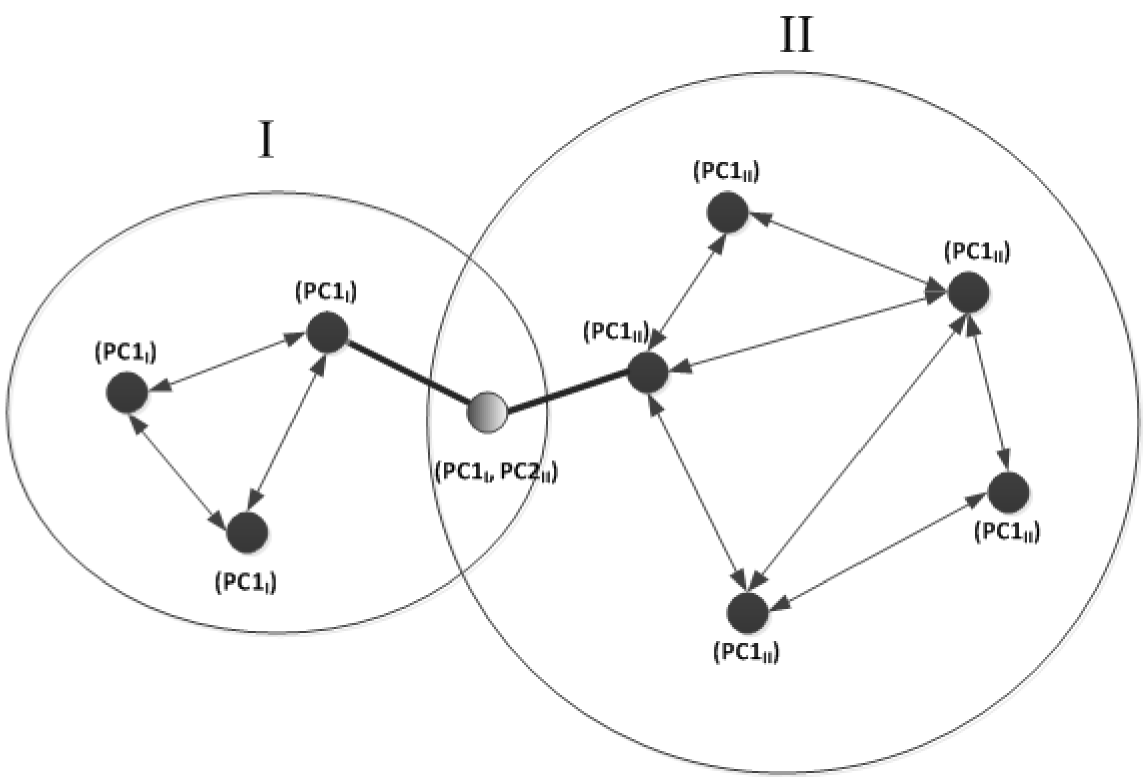


Figure 4.4: Limited merging between two networks, I and II, with one node in possession of $PC1_I$ and $PC2_{II}$ acting as gateway between the networks.

In order for this limited merging to be of any value for the networks, the node needs to be able to broadcast the network prefix of each network, becoming a gateway node, to each other so that nodes in both networks can communicate with each other through the gateway node. It is important to notice that nodes in one network will not get routing updates from the other network, and they will not learn about each other before another entity takes care of this information. For instance, a special node could get a PC1 from every network on site using e.g. out-of-band authentication and then become a DNS server for the site. We wont go into more detail on this because that is not part of our scope, but can be interesting to note.

This functionality may be very beneficial for some parties, but too insecure for others - therefore we note that this functionality needs to be optional for the users. For example, in a large emergency situation a few actors like civilian paramedics and The Red Cross personnel might want this functionality, but the military might want to opt out to protect their secure network.

4.7.2 Full Merging

The limited merged network explained above does not scale very well, and should only be used temporarily until the networks can be fully merged. For the networks to be fully merged there needs to be some trust established between the different service proxies managing the networks.

If the SPs are able to verify each others as trusted SPs based each others LLPKC, then each SP can sign each other a PC0 for each others network. This way, they both become SPs in a large fully merged network. If they are not able to reach each other, or not verify each others LLPKC they should be able to use an out-of-band authentication. As before, we will not go further into how this is done, but it is worth mentioning.

After the SPs have signed each others PC0, they will send their ALs to each other in order to make a full AL for the whole fully merged network. After this point, all nodes in all merged networks will be able to verify each other and from the point of view of the regular nodes the network merging is complete.

The SPs however, needs to decide upon a master SP which will take care of new unauthenticated nodes and broadcasting the full AL. The choice might be different based on the requirements of the users. I.e. it might be done in a arbitrarily fashion, or it might be determined out-of-band based on some real-world characteristics.

4.8 Assumptions and Limitations

Some assumptions have been made to make our system design possible. These are explained here in this section as well as potential limitations these assumptions may introduce.

4.8.1 Pre-Configured Long-Lived Public Key Certificates

In our system design we have assumed that before a node can be issued a PC1 and become a fully trusted node, it is obligated to present a valid LLPKC to the SP managing that network. This means that all nodes must be given a LLPKC signed by a CA at some point to be able to connect to a restricted network. This certificate could either be manually pre-configured or perhaps given to the node out-of-band sometime.

However, even though a node is in possession of a LLPKC it might still not be recognized by the current SP. In that case the node will only be issued a PC2 until it eventually gets a certificate which is valid. This can be given to the node through some out-of-band authorization which leads to the SP signing the node a new LLPKC, and revoking it when the scenario is over. Thus the node would get full access to the network for a limited time.

If a node is not in possession of any LLPKC at all, it is still able to be part of the network with the help of a PC2. This way we assure that the network is aware of nodes that may potentially be important and should consider being issued a PC1 or LLPKC trough the network or out-of-band.

However, this is something that could be up to the SP to decide. If it finds it necessary to establish a network without validating LLPKC, it could issue PC1 to all nodes who want to participate in a network. The SP is then creating a network that is restricted in the sense that only nodes in the network are able to communicate with each other and the only way to get an certificate would be through the SP.

The reason for why we use LLPKC in our system design is in order to do a full authentication of nodes. That is, with a LLPKC you are able to have a unique relation between an identity and its certificate. It is therefore a good way of providing a verification of an identity, while the proxy certificates are primarily used for verifying that the node in possession of it have access to the network, it can be indifferent of the real identity of the node given the settings of the SP.

4.8.2 Service Proxies - Operational Command Centers

The Service Proxy is a central entity which is in charge of the access control in a network. It is argued that central nodes are not preferred in ad hoc network as they violate the goal of a truly decentralized network. However, to be able to have a proper authentication scheme in our system, it is necessary to have some central nodes that can handle the access control to the network. These task of a central node can tied to the responsible operational center on the site, as described in Chapter 3.

4.8.3 Valid LLPKCs

During the authentication handshakes, the SP might not have access to the Internet and thus not able to get the latest copies of the Certificate Revocation Lists (CRL). This means that even though an SP is able to verify nodes LLPKC, it might have been marked as invalid/revoked in the CRL. We assume however that the SP will authenticate new nodes based on the knowledge it has at the moment of authentication, and rather check with the CRLs after an Internet gateway has been set up.

4.8.4 Multi-Layered Security

Security is a multi-layered issue and we consider only routing done on the network-layer in our system design. We assume users of our protocols use proper transport- and application-layer security protocols. The proxy certificates used in our system is meant for routing-security, and we assume they are not used for security on the layers above. We therefore assume our PCs and LLPKCs are used for hardware authentication, not user authentication which would be handled upper layers.

4.8.5 IP Allocation

When a new node is discovered, it would probably have a link-local IP address which in turn requires that an address allocation scheme must be in place, e.g. DHCP. This responsibility would be natural to assign to the SP, but depending on the network configuration other nodes could also be in charge of this. In our scheme we do not however cover this setup, and we use static IP allocation in our implementation explained in Chapter 5.

4.9 Security Considerations and Issues

In this section we cover some of security considerations and issues that is worth mentioning.

4.9.1 Public Keys

One of the security features of our system is that the public keys belonging to the PC0s, PC1s and the SPs LLPKCs are only public inside the network of fully trusted nodes. This enables us to use the private key of the PC1s belonging to trusted nodes and LLPKC to the SP to encrypt the authentication lists broadcasted thus the lists will only be readable for nodes inside the network.

However, using the private key for signing and encrypting is not recommended as this might expose the key to potential cryptanalytic attacks. But for the SP it only uses the private key of its LLPKC to self-sign its own PC0, thus not exposing its key in such degree.

4.9.2 Security Issues

Being based on a wireless network, our system is vulnerable to security attacks that take advantage of this shared medium. Attacks such as Denial of Service (DoS), ”man-in-the-middle” and jamming can severely paralyze and damage a communications network and can be very hard to avoid [PLH06]. Protecting our system against these kinds of attacks would be challenging.

Chapter 5

Implementation

Due to the time constraints during this project we where by far able to implement the full functionality behind the system design as explained in Chapter 4. We have however focused on implementing a solution that at least show the principles behind how the authentication process between nodes should be performed.

The first section in this chapter describes shortly some of the main changes that have been done in the implementation compared to what is described in the system design. It then continues to explain the source code we have changed to incorporate our solution and finally gives a detailed description of how our implementation works.

5.1 Modifications and Restrictions

Some of the restrictions that has come to be in our implementation is that no certificates are actually sent during an authentication process and no signatures are appended when authenticated nodes are communicating routing information in the restricted network. These signatures have been replaced with an integer value that is agreed upon during a 4-way handshake between two nodes. After the nodes agree upon a common value, they will append these values when sending routing information thus only accepting and processing messages that contain the same value.

Because no nodes are in possession of any certificates there will not be any node that is assigned to take the responsibility of acting as the Service Proxy (SP). However, this role is something that is assigned during the first 4-way handshake when establishing a network which will be further explained in section 5.3.2. After two nodes establish a network together, the node who has become SP will be the only one capable of performing a 4-way handshake with new unauthenticated nodes entering the network. We will refer to the node with the role of SP as master node because it it does not have the same power as a true SP hence the name is misleading.

In our system design we make three distinct roles, making every trusted node able to connect new nodes to the network. In this implementation however, there is only two roles - master and authenticated. Because of this simplification it is not possible to add new nodes that are not direct neighbors with the master.

It is important to mention that this algorithm will serve as a proof of concept, rather than an actual authentication algorithm. We aim to show that the BATMAN protocol can be extended to the system described in the previous chapter, but this implementation does only the very groundwork for this extension.

5.2 B.A.T.M.A.N. Daemon

We used the BATMAN Daemon (batmand) version 0.3.2 source code as a base for our implementation. The source code can be downloaded from http://www.open-mesh.org/ which is web-page where you can find a collection of tools to build free and open mesh networks. The page contains everything regarding batmand and the current development on a different version of BATMAN called BATMAN Advanced (batman-adv).

Batmand is a network layer routing protocol utilizing UDP datagrams to send the OGMs, while the batman-adv operates on the link-layer. Both versions manipulate the routing table in the Linux kernel based on the OGMs sent in the network, such that applications above the network-layer can exchange messages over the BATMAN-network.

We chose batmand over batman-adv, because being a network layer protocol, it has far simpler functionality and should in theory be easier to extend and manipulate. As it only uses UDP datagrams to send data over the network interface, we don't need to worry about different types of link layer datagrams either.

5.3 Implementation Description

Here we explain more detailed how our implementation works. Different entities and functionalities are described and tied up to concepts from our system design. Our contributions to the source code can be found in Appendix C.

5.3.1 Roles

The authentication process in our implementation is built around the aspect of giving the participating nodes specific roles in the network. A node starts with the role unauthenticated, but can be assigned the role authenticated or master which is represented in the code with a role variable set to 0, 1, or 2 respectively. How this role is assigned is described in detail in the sections below.

The roles can be seen as simplifications of the PCs used in our system design. A master node with role 2 is equivalent to a node in possession of a PC0, and the authenticated node with role 1 is equivalent to a node with PC1. However, a node with role 0 is not the same as a node with PC2. Even though neither has verified their identity through a proper authentication process, a PC2 can be part of a established network with limited rights, while the node with role 0 will have no rights, and can therefore not process or rebroadcast other OGMs. This design choice is done mainly because of simplicity.

5.3.2 4-way Handshake

The 4-way handshake consists of sending in total 4 messages between two nodes. These messages are modified OGMs with three values defining the different steps during the handshake - namely challenge, challenge-response and authentication. A complete 4-way handshake is shown in Figure 5.1.

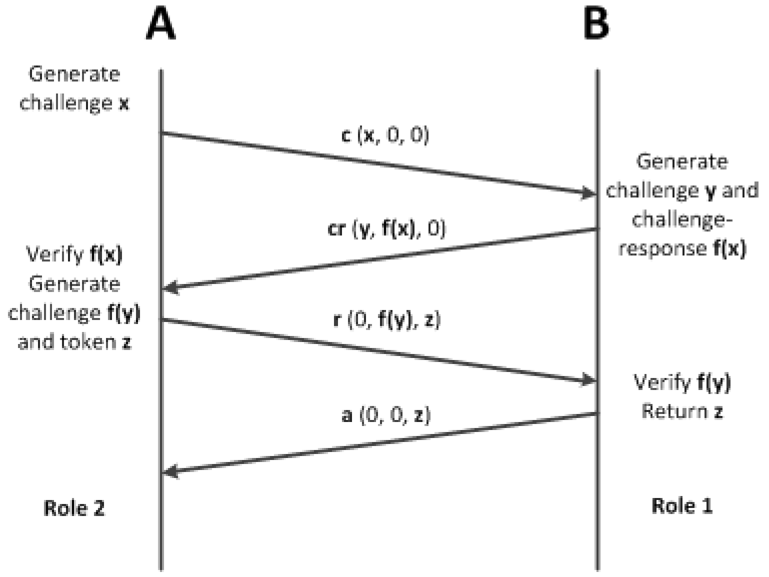


Figure 5.1: A complete 4-way handshake

The first step in the 4-way handshake is to send an OGM with a challenge value which is adequately named Challenge message. This challenge value corresponds to the nonce value in message II in the authentication handshake, as proposed in Section 4.2.2.

The node receiving the Challenge message will generate and send a Challenge-Response message with a new challenge value and the response value set as a function of the challenge received, which corresponds to message III in the authentication handshake. The function used here corresponds to the encrypted hash-function of the nonce.

If the receiving node verifies the response, it returns a Response message where the response value set as a function on the received challenge in the Challenge-Response message. In addition the message is appended an authentication value to be used as the authentication token in the network. These messages loosely corresponds to message IV and VI in 4.2.2, except that the authentication value sent is going to be used as a signature later, whereas the proxy certificate sent in message VI is used to make individual signatures.

The node that sent the Response message now waits for the other node to send an OGM with the same authentication value as a confirmation that he has received the token. If the correct message is received the 4-way handshake is complete and the authentication value, will be saved and used in every OGM from this point on and we refer to this only as OGM messages. The same node also sets his role to two, becoming the master node, given it was not the master from the very outset.

The other node, upon reception of the Response message, also saves this authentication token value, and sets its role to one, becoming an authenticated node.

5.3.3 Detailed Authentication Description

The 4-way handshake is triggered whenever an unauthenticated node or the master node receives an OGM with the challenge, response, and authentication values set to zero. These messages we call Plain OGMs.

The challenge values are always set by generating a numbers with a pseudo-random number generator (PRNG). The function used to generate response values is very simple since it multiplies the challenge by two modulo of the largest possible number. For our actual system design these functions would be replaced with a stronger cryptographic algorithm based on public key cryptography.

The master node will not accept any messages from nodes authenticated in other networks. That is, messages that contain an authentication value greater than zero, but different from the masters token will be discarded. The master node will also drop any Challenge-Response OGMs it may receive.

For simplicity, we have chosen to let all authenticated nodes discard messages with a different or zero values in the authentication token. This means that new nodes have to be direct neighbor of the master node to be able to authenticate themselves.

Random Back-off

Because we have not implemented a stand-alone authentication protocol using a separate socket, we added our handshake values to the OGMs being scheduled by the BATMAN protocol. We where reluctant to alter the message flow as this would change the mode of operation of the whole BATMAN OGM scheduling which might cause unintentional consequences for the routing.

We quickly realized that when two unauthenticated nodes discover each other with Plain OGMs, a 4-way handshake will be triggered at both nodes approximately at the same time. For this reason we included a random back-off function which makes a node back off from sending Challenges when a handshake collision is detected. That is, if a node sends a challenge, but receives a challenge before the handshake is complete, the node will generate a random back-off time of less than 10 seconds.

The back-off function does not halt other operations of BATMAN, it only clears all challenge and response values and make sure no new challenge is to be sent before the random back-off time expired. If the node receives a Challenge before his back-off time is expired, he replies to the as usual with a Challenge-Response.

However, this back-off time does not completely solve the problem. After a node has detected a collision and sets its random back-off time, it will still continue to receive at least a few more identical Challenges because of the high frequency in which the OGMs are scheduled and sent at the other node. These Challenges should also be discarded thus we made a quick fix by not allowing any Challenges to be processed before half the random back-off time has passed. Figure 5.2 illustrates this problem. The red field in the figure indicates the period in which no challenges are accepted after a collision.

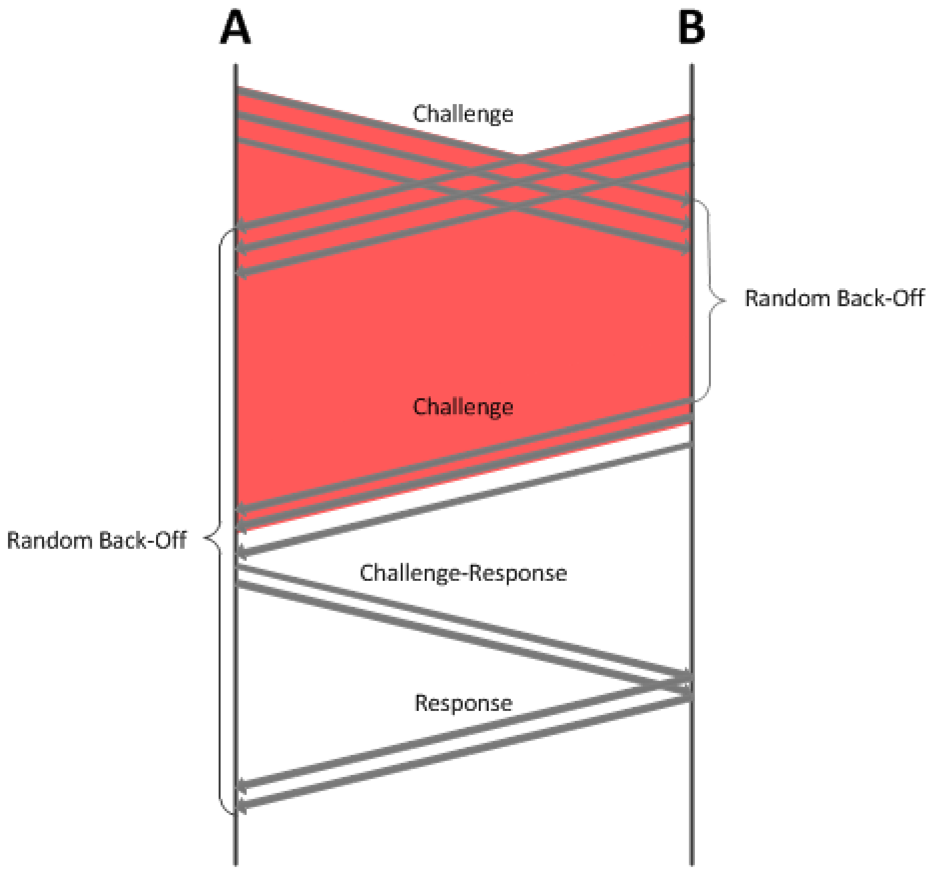


Figure 5.2: Back-off in the handshake algorithm. Node A only accepts Bs challenges halfway through the random back-off time

This is however not a very efficient way of dealing with the problem and creates unnecessary delay in the authentication process. But if the functionality is moved into separate authentication protocol this could be avoided. We have decided to do this with the handshakes in our system design, see 4.3.2. We will however test our implementation and get some numbers which we can compare.

Chapter 6

Laboratory Environment and Testing

This chapter will show how our laboratory environment was set up, what equipment was utilized and the how the actual testing was performed.

6.1 Laboratory Environment

A small laboratory environment was set up in an indoor office area. Because the space was limited, we had to come up with some untraditional methods in order to get the network into the necessary states.

6.1.1 Computer and Network Setup

We had four computers in total at our disposal during our project. All of the computers were running Linux and had both our implementation and the original BATMAN protocol installed. We refer to these lab computers as BATMAN nodes or just nodes out this chapter. The BATMAN nodes built the ad hoc network between each other using their wireless interfaces, while their Ethernet interface was used as a control channel for our workstations to be able to control the nodes and log the activity.

6.2 Testing

The goal of the testing was to compare how our implementation performs compared to the original BATMAN protocol. One of the most important indicators of how well a routing protocol performs, is its convergence time. This is a measure which shows how fast every node in the network is aware of a change in the networks topology, such as a loss of an active link. Another factor that we know put a significant delay on our modified protocol, was the 4-way handshake performed when authenticating nodes.

It is therefore natural to test the protocol in scenarios that affect these parameters. Table 6.1 shows a summary of the test scenarios that will be run using both the modified and the original version of the BATMAN protocol.

Test	Description
I	Authentication time; two unauthenticated nodes
II	Authentication time; unauthenticated node enters a network
III	Convergence time; node disappears
IV	Convergence time; previously authenticated node rejoins the network
V	Convergence time; unauthenticated node enters before master rejoins the network

Table 6.1: Description of different tests to be performed on our small testbed

To explain the test scenarios in further detail we first define the values that will be used as results and compared against each other after the testing.

Authentication time: Time taken for a unauthenticated node to be authenticated by either another node or by a master node.

Convergence time: Time taken for the nodes in the network to recognize that a route from a source to a sink node is down after intentionally disabling an active link in the path.

6.2.1 Testing Procedure

In all the testing scenarios, we used our workstations to control the nodes in the test bed via Secure Shell (SSH) over the Ethernet network. To start running the BATMAN protocol on a node, we made a small run bash script which first starts the BATMAN daemon and then opens an interface on the UNIX socket enabling debugging of the running protocol. See Appendix B for more details about the script and the how the nodes were setup.

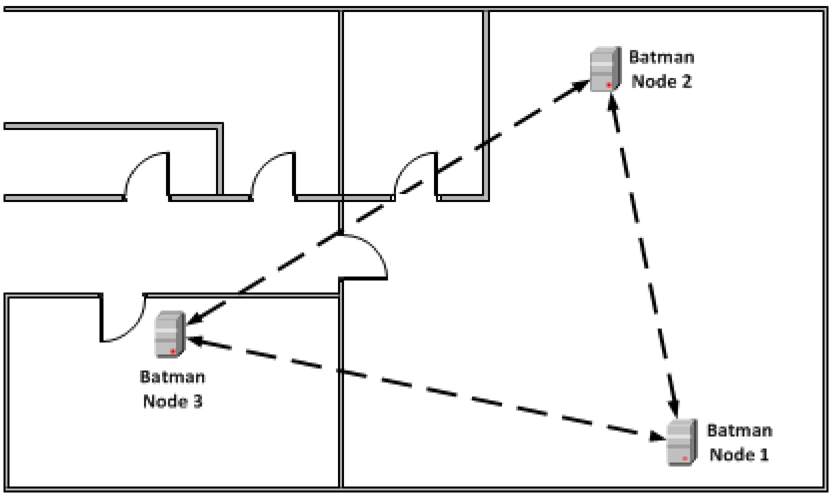


Figure 6.1: Setup for Test I and II.

Test I The BATMAN daemon was started on two nodes, node 1 and 2 as shown in Figure 6.1. They where left running until they completed their 4-way handshake and had added each other to their routing tables. The authentication time we used here was the time from when the first OGM is received by the node who becomes the master, until this node has added the other in its routing protocol.

Test II Two nodes were started like in Test 1. After the network was established and had stabilized, the BATMAN daemon was started on a new node which was then introduced to the network, i.e. node 3 in Figure 6.1. We stopped the test when the new node had been added to the master nodes routing table. The authentication time was the time measured from when the master node received the first OGM from the unauthenticated node until it added the node to its routing protocol.

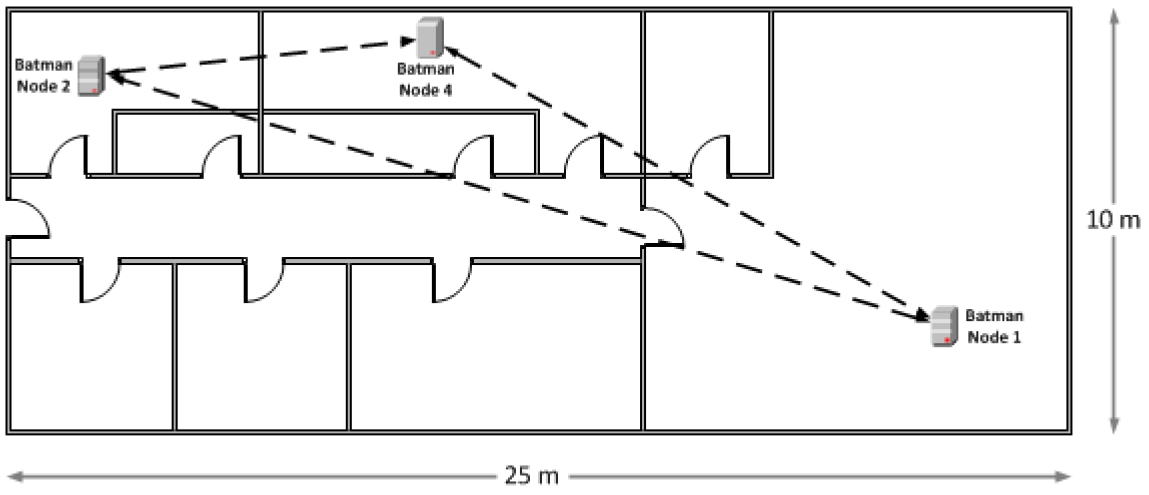


Figure 6.2: Setup for Test III and IV. Node 1 was the master node and the source, and node 2 the sink.

Test III A network was established between all of the three nodes as shown in Figure 6.2. When the network was stabilized, node 4 was removed from the preferred path between the source (1) and the sink (2) node.

When the source node updated its preferred route to the sink, the test was stopped. The convergence time was measured from when the source node last received a broadcast from node 4, until the source node updated its routing tables accordingly.

In order to force the network into believing that the best paths were through node 4, we had to reduce the transmitting power of the source and sink node to 7dBm. To make node 4 disappear and rejoin without re-authentication, we had to set the authentication token value manually in node 4s code, so we could kill the process and restart it bypassing the authentication handshake.

Test IV When the routing tables had stabilized after the previous test, we reintroduced node 4 to the network. Now we measured the time difference from when the source node first received an OGM from the rejoined node until that node became a part of his preferred route to the sink node again.

Test V In this test a network of three nodes was established. Once established, the master node was removed and a new unauthenticated node was introduced to the network. After the network had stabilized, we reintroduced the master node and let it start the handshake with the new node. The time measured here was the difference from when the master node first receives an OGM from the network, to the point where the master node adds the new node to its routing tables.

Unfortunately, we had problems performing the test V because of time, equipment and laboratory limitations. We were not able to produce several distinct routes with more than three nodes, or else the two different routes would be so much the same that convergence would be in a matter of one or two seconds and not realistic to any real world scenario. And for test V we needed a minimum of four nodes - one sink, one source, one master and one regular node for which both constitutes different routes between the source and the sink. With more time we could have been able to find another way of performing the test.

The timestamps used for calculating the authentication and convergence time is found from analyzing the debug output generated while running the BATMAN protocol. The complete logs from the testing can be found in Appendix D. The results from these tests can be found in the next chapter.

Chapter 7

Results

This section presents the numeric results collected from the tests described in the previous section. The results can be divided in two sections, one for authentication time and one for convergence time.

7.1 Authentication Time

The authentication times calculated for Test I and Test II are shown in Table 7.1 along with the minimum, maximum, and average time values.

#	**Test I**	**Test II**
1	13.30	5.82
2	17.39	9.51
3	8.03	15.24
4	15.46	19.10
5	11.21	19.72
6	12.16	7.07
7	9.13	31.03
8	16.33	30.53
9	17.10	14.10
10	13.99	7.28
Min	8.03	5.82
Max	17.39	31.03
Avg	13.41	15.94

Table 7.1: Results after 10 runs with Test I and Test II

The original BATMAN protocol does of course not have any authentication mechanism thus the first two tests was only run with the modified BATMAN protocol.

As we can see from the results above, the authentication process adds significant overhead in the initial phase when establishing new networks or authenticating new nodes. The large difference between the best and worst case can simply be explained as randomness. It is arbitrary how many collisions will occur during an authentication process and size of the back-off time could in the worst case be set to a very high value as this is chosen randomly.

7.2 Convergence Time

Here we aim to measure route convergence time in three different scenarios and see how our implementation performed against the original BATMAN protocol. Our aim here is mainly to show that once authenticated, BATMAN operation should perform similar to that of the original protocol. Table 7.2 shows the results for test III and IV.

	Test III		**Test IV**	
#	**Secure**	**Original**	**Secure**	**Original**
1	18.26	20.91	41.40	29.85
2	15.02	22.14	43.04	44.02
3	16.00	14.22	33.23	41.80
4	14.75	13.07	48.46	44.38
5	17.94	17.45	45.82	47.28
Min	14.75	13.07	33.23	29.85
Max	18.26	22.14	48.46	47.28
Avg	16.39	17.56	42.39	41.47

Table 7.2: Results after 5 runs with Test III and Test IV with both our implementation and the original BATMAN protocol

With this table we can produce far more positive results. When it comes to convergence within the network between already authenticated nodes, our implementation does not seem to add any delay. However, as we can see, neither our implementation nor the original implementation performs great. This coincides with previous tests which we mentioned in 2.2.3.

Chapter 8

Conclusion

In this project we have proposed a solution where we combine common security measures to provide a restricted ad hoc network. Public key certificates are used as a means for access control to a network and they are managed by semi-central nodes. The proposal solves the task of issuing certificates by giving trusted nodes rights on behalf of the semi-central nodes to include new nodes to the network, and to be able to verify other trusted nodes without the presence of the semi-central nodes. We accomplish this without significantly affecting the security or the performance of the network.

This project started with a comprehensive study of ad hoc networks and their characteristics that was important to consider when trying to find a solution. We also researched the necessary background information about the tools we planned to use - evaluating some routing protocols and different cryptographic schemes that seemed fit in an ad hoc environment.

Along with this research it was important to consider the environments and scenarios the ad hoc network was to be used as they would probably influence the network behavior.

We used an ad hoc routing protocol called BATMAN as the base for our entire system. The protocol is simple and robust with limited complex functionality compared to many of its alternatives - making it easier to modify for security purposes and to verify its security. The proposed solution adds several new entities to the original BATMAN protocol, which is used in order to achieve a scalable authentication scheme.

The proposed system incorporates access control without being dependent on a central authority, rather some entity taking charge at the place and time of the network initialization. We also focused on having the network being able to continue its operation and partially allowing new entities and fully verifying already trusted entities even if the entity with the master role is unavailable. Even better, several entities can mirror the master role making the authentication scheme very scalable.

For full authentication our system either requires dependence on a central authority or an out-of-band authentication taking place on site, both of which might be impractical for ad hoc networks. However, we argue that this is required in order to have proper authentication. When establishing a network this might be optional for the users, and will most likely be depending on the scenario. Either way, our system supports both settings.

A simple implementation was developed as a proof of concept and tested to indicate the impact the additional functionality has on the network. It is in no way a secure implementation, but we have modified the protocol in such a way that it could eventually be extended to incorporate the complete system.

The results from the two first performance tests seemed to prove our hypothesis that our handshake process adds significantly time overhead during establishment. Our choice of using a random master node allocation seems to be a poor choice and separating the pre-authentication steps from routing messages is definitely recommended - both have been taken into consideration in our proposed system design.

We performed two additional tests with far more promising results. We assumed that our implementation would not influence regular BATMAN operation, and tested this assumption by checking route convergence times for already authenticated entities and compared the results with the original BATMAN protocol. We did not discover any discrepancies between the two setups which implies that the development of our implementation should be continued.

Our system design accomplishes to provide a proposal for a restricted ad hoc network that takes into account the challenges of their unique nature. Our implementation shows that the BATMAN protocol is a good choice to utilize when trying to implement our solution and that the work done in this project is a step in the right direction.

8.1 Further Work

This section suggests some topics and areas that would be interesting to investigate further. These topics are omitted due to time constraints, but they important and needs to be looked at later on.

A simulation of our system design is recommended in order to investigate how the networks would behave in larger scale and rapidly changing environments. Based on the results from the simulations, changes could be made to the design. It would also be desirable to continue the implementation of our solution and to do some real life tests to see how the system would behave.

8.1.1 In-depth Security Analysis

The design should be tested with some gray box security analysis, i.e. analyzing the specifics of our implementation, not the cryptographic functions that will be used. In addition, a cryptanalysis of the actual implementation would also possibly strengthen the protocols claims.

Bibliography

[CJ10] Thomas Heide Clausen and Philippe Jacquet. Optimized Link State Routing Protocol (OLSR). *Network Working Group*, Last accessed December 19, 2010. http://tools.ietf.org/html/rfc3626.

[EA04] Jon Edney and William A. Arbaugh. *Real 802.11 Security: Wi-Fi Protected Access and 802.11 i*. Addison Wesley Publishing Company, 2004.

[FHT10] Stephen Farrell, Russ Housley, and Sean Turner. An Internet Attribute Certificate Profile for Authorization. Last accessed December 20, 2010. http://tools.ietf.org/html/rfc5755.

[HS98] Yung-Kao Hsu and Stephen Seymour. Intranet security framework based on short-lived certificates. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1997., Proceedings Sixth IEEE workshops on*, pages 228 –234, 1998.

[IEE07] IEEE Standards Association. IEEE Std. 802.11-2007, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std. 802.11, 2007*, June 2007.

[KR09] James Kurose and Keith Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Pearson Education, 5rd edition, 2009.

[Mes10a] Open Mesh. *B.A.T.M.A.N. V status update*. *open-mesh.org*, Last accessed december 19, 2010. http://www.open-mesh.org/wiki/2010-12-12-batman-v-status-update.

[Mes10b] Open Mesh. *Why starting B.A.T.M.A.N.?* *open-mesh.org*, Last accessed december 19, 2010. http://www.open-mesh.org/wiki/why-starting-batman.

[MM04] C. Siva Ram Murthy and B. S. Manoj. *Ad Hoc Wireless Networks: Architectures and Protocols*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2004.

[NALW10] Alex Neumann, Corinna Aichele, Marek Lindner, and Simon Wunderlich. Better Approach To Ad-Hoc Networking (B.A.T.M.A.N) draft-wunderlich-open-mesh-manet-routing-00. *Network Working Group*, Last accessed December 19, 2010. http://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00.

[PLH06] Al-Sakib Khan Pathan, Hyung-Woo Lee, and Choong Seon Hong. Security in wireless sensor networks: Issues and cshallenges. In *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, pages 6 pp. –1048, 2006.

[Sta06] William Stallings. *Cryptography and Network Security: Principles and Practice.* Prentice Hall, 4th edition, 2006.

[TET+10] Steven Tuecke, Doug Engert, Mary Thompson, Laura Pearlman, and Von Welch. Internet X.509 Public Key Infrastructure Proxy Certificate Profile. *Network Working Group*, Last accessed December 20, 2010. http://tools.ietf.org/html/rfc3820.

[WSK06] Eli Winjum, Pål Spilling, and Øyvind Kure. *Ad Hoc networks used in emergency networks : the Trust Metric Routing approach.* FFI Rapport, 2006.

[YLY+04] Hao Yang, Haiyun Luo, Fan Ye, Songwu Lu, and Lixia Zhang. Security in mobile ad hoc networks: Challenges and solutions. *Wireless Communications, IEEE*, pages 38 – 47, 2004.

[ZH99] Lidong Zhou and Zygmunt J. Haas. Securing Ad Hoc Networks. *Network, IEEE*, pages 24 –30, 1999.

[ZLdCG08] Huzaifa Zafar, Victor Lesser, daniel Corkill, and Deepak Ganesan. Using Organization Knowledge to Improve Routing Performance in Wireless Multi-Agent Networks. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 821–828. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

Appendix A

BATMAN Protocol

This appendix contains some additional details about the BATMAN ad hoc routing protocol.

A.1 Originator Message (OGM) Format

The core algorithm in the BATMAN protocol as well as its implementation have gone through several evolutionary changes during development. Currently the BATMAN developers are working on a version V of the protocol where they aim to improve issues such as mesh bonding, weighted Link Quality (LQ) measurements and multicast optimizations [Mes10a].

During development there has also been some changes to the Originator Message (OGM) format. Figure 2.2 shown in Section 2.2.2 is the OGM as it is described in the Internet-Draft [NALW10]. Two fields, Previous Sender Address and Transmit Quality (TQ), have been added to the OGM as shown in Figure A.1.

A BATMAN packet consisting of an Originator Message (OGM) together with zero or more HNA extension messages, is encapsulated in a single UDP data packet. The format of the OGM is shown in Figure A.1.

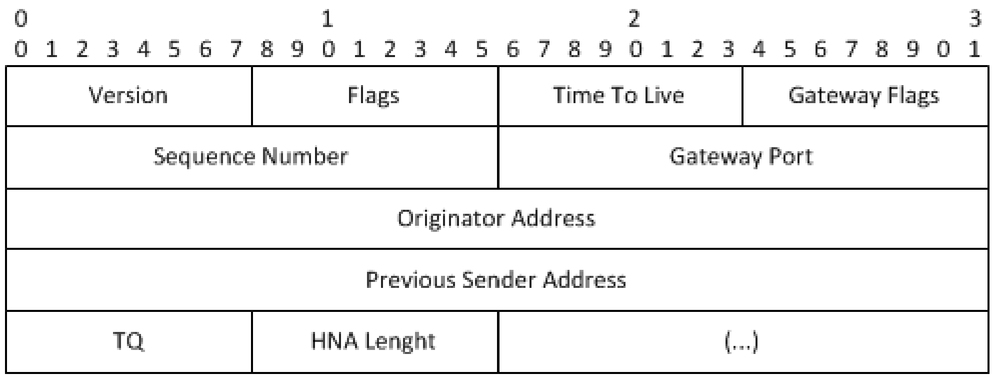


Figure A.1: Originator Message (OGM) Format.

The different fields in the OGM is explained below:

Version
Identifies the version of BATMAN for the contained message

Is-direct-link flag
Flag indicating whether a node is a direct neighbor or not.

Unidirectional flag
Flag indicating whether the neighboring node is bidirectional or not.

Time To Live
Contains the maximum number of hops a message will be transmitted.

Gateway Flags
Indicates whether a node may act as a gateway with access to the Internet.

Sequence Number
Number added by an Originator to every OGM it broadcasts. Number is incremented for each OGM broadcasted.

Originator Address
The IPv4 address of the B.A.T.M.A.N. interface on which behalf the OGM has been generated.

A.1.1 Host Network Annoncement Message Format

The Host Network Announcement (HNA) is used to announce that a node is a gateway to another network. If so, the node sets the Gateway Flag in the OGM and appends a HNA-extension-message containing the netmask and the network address of the announced network.

HNA-extension-message format is shown in Figure A.2.

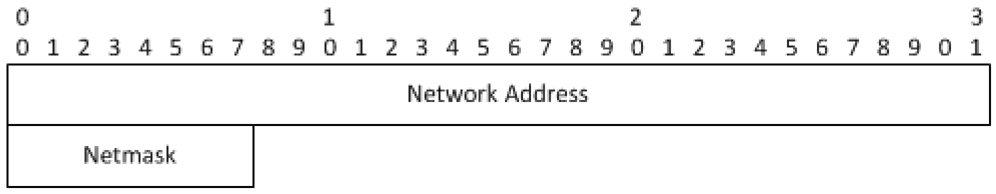


Figure A.2: Host Network Announcement Message format.

Appendix B

Lab Setup

The computers used in the lab was setup with the following hardware:

- Intel Core 2 Duo 2.83 GHz processor
- 4 GB memory
- Atheros AR5413 802.11abg NIC

Further, they are setup with Ubuntu 10.4 (Linux Kernel 2.6.32-25-generic-pae) and ath5k drivers for the wireless interfaces. The network interface is configured as follows:

```
/etc/network/interfaces

auto lo
iface lo inet loopback

auto wlan0
iface wlan0 inet static
address 10.0.0.X
netmask 255.255.255.0
pre-up ifconfig wlan0 down
pre-up ifconfig wlan0 hw ether XX:XX:XX:XX:XX:XX
pre-up iwconfig wlan0 mode ad-hoc essid BATMAN channel 3

auto unicast
iface unicast inet static
address 10.0.0.X
netmask 255.255.255.0
pre-up brctl addbr unicast
pre-up brctl addif unicast wlan0
pre-down ifconfig unicast down
post-down brctl delif unicast wlan0
post-down brctl delbr unicast
```

To install batmand, run the following as root user:

```
make
make install
make clean
```

For running the batman daemon on the test nodes, we ran the following script as root user:

```
ifconfig wlan0 up
batmand wlan0
batmand -cd 4
killall batmand
ifconfig wlan0 down
```

For reducing the transmitting power in test III and IV we ran the following as root user:

```
ifconfig wlan0 down
iwconfig wlan0 txpower 7
ifconfig wlan0 up
```

Appendix C

Source Code

C.1 BATMAN

C.1.1 batman.h - struct bat_packet

```
struct bat_packet
{
  uint8_t  version;  /* batman version field */
  uint8_t  flags;    /* 0x80: UNIDIRECTIONAL link, 0x40: DIRECTLINK flag,
     ... */
  uint8_t  ttl;
  uint8_t  gwflags;  /* flags related to gateway functions: gateway class */
  uint16_t seqno;
  uint16_t gwport;
  uint32_t orig;
  uint32_t prev_sender;
  uint8_t tq;
  uint8_t hna_len;
  //ENOTE: challenge=0 -> "no challenge".
  uint8_t challenge;
  //ENOTE: response=0 -> "no response".
  uint8_t response;
  //ENOTE: auth_token=0 -> "not authenticated".
  uint8_t auth_token;
} __attribute__((packed));
```

C.1.2 batman.c - batman()

```
int8_t batman(void)
{
  struct list_head *list_pos, *forw_pos_tmp;
  struct orig_node *orig_neigh_node, *orig_node;
  struct batman_if *batman_if, *if_incoming;
  struct forw_node *forw_node;
  struct bat_packet *bat_packet;
  uint32_t neigh, debug_timeout, vis_timeout, select_timeout, curr_time;
  unsigned char in[2001], *hna_recv_buff;
  char orig_str[ADDR_STR_LEN], neigh_str[ADDR_STR_LEN], ifaddr_str[
     ADDR_STR_LEN], prev_sender_str[ADDR_STR_LEN];
  int16_t hna_buff_len, packet_len, curr_packet_len;
  uint8_t forward_old, if_rp_filter_all_old, if_rp_filter_default_old,
     if_send_redirects_all_old, if_send_redirects_default_old;
```

```
uint8_t is_my_addr, is_my_orig, is_my_oldorig, is_broadcast, is_duplicate,
     is_bidirectional, has_directlink_flag;
int8_t res;

debug_timeout = vis_timeout = get_time_msec();

if ( NULL == ( orig_hash = hash_new( 128, compare_orig, choose_orig ) ) )
  return(-1);

/* for profiling the functions */
prof_init(PROF_choose_gw, "choose_gw");
prof_init(PROF_update_routes, "update_routes");
prof_init(PROF_update_gw_list, "update_gw_list");
prof_init(PROF_is_duplicate, "isDuplicate");
prof_init(PROF_get_orig_node, "get_orig_node");
prof_init(PROF_update_originator, "update_orig");
prof_init(PROF_purge_originator, "purge_orig");
prof_init(PROF_schedule_forward_packet, "schedule_forward_packet");
prof_init(PROF_send_outstanding_packets, "send_outstanding_packets");

list_for_each(list_pos, &if_list) {
  batman_if = list_entry(list_pos, struct batman_if, list);

  batman_if->out.version = COMPAT_VERSION;
  batman_if->out.flags = 0x00;
  batman_if->out.ttl = (batman_if->if_num > 0 ? 2 : TTL);
  batman_if->out.gwflags = (batman_if->if_num > 0 ? 0 : gateway_class);
  batman_if->out.seqno = 1;
  batman_if->out.gwport = htons(GW_PORT);
  batman_if->out.tq = TQ_MAX_VALUE;

  schedule_own_packet(batman_if);
}

if_rp_filter_all_old = get_rp_filter("all");
if_rp_filter_default_old = get_rp_filter("default");

if_send_redirects_all_old = get_send_redirects("all");
if_send_redirects_default_old = get_send_redirects("default");

set_rp_filter(0, "all");
set_rp_filter(0, "default");

set_send_redirects(0, "all");
set_send_redirects(0, "default");

forward_old = get_forwarding();
set_forwarding(1);

while (!is_aborted()) {

  debug_output( 4, " \n" );

  /* harden select_timeout against sudden time change (e.g. ntpdate) */
  curr_time = get_time_msec();
  select_timeout = ((int)(((struct forw_node *)forw_list.next)->send_time
      - curr_time) > 0 ?
        ((struct forw_node *)forw_list.next)->send_time - curr_time : 10);
  res = receive_packet(in, sizeof(in), &packet_len, &neigh, select_timeout
      , &if_incoming);

  /* on receive error the interface is deactivated in receive_packet() */

  if (res < 1)
    goto send_packets;

  curr_time = get_time_msec();
  curr_packet_len = 0;
  bat_packet = (struct bat_packet *)in;

  addr_to_string(neigh, neigh_str, sizeof(neigh_str));
  addr_to_string(if_incoming->addr.sin_addr.s_addr, ifaddr_str, sizeof(
      ifaddr_str));

  while ((curr_packet_len + (int)sizeof(struct bat_packet) <= packet_len)
     &&
    (curr_packet_len + (int)sizeof(struct bat_packet) + bat_packet->
        hna_len * 5 <= packet_len) &&
    (curr_packet_len + (int)sizeof(struct bat_packet) + bat_packet->
        hna_len * 5 <= MAX_AGGREGATION_BYTES)) {

    bat_packet = (struct bat_packet *)(in + curr_packet_len);
    curr_packet_len += sizeof(struct bat_packet) + bat_packet->hna_len *
        5;

    /* network to host order for our 16bit seqno */
    bat_packet->seqno = ntohs(bat_packet->seqno);

    addr_to_string(bat_packet->orig, orig_str, sizeof(orig_str));
    addr_to_string(bat_packet->prev_sender, prev_sender_str, sizeof(
        prev_sender_str));


    is_my_addr = is_my_orig = is_my_oldorig = is_broadcast = 0;

    has_directlink_flag = (bat_packet->flags & DIRECTLINK ? 1 : 0);

    debug_output(4, "Received BATMAN packet via NB: %s, IF: %s %s (from OG
        : %s, via old OG: %s, seqno %d, tq %d, TTL %d, V %d, IDF %d) \n",
        neigh_str, if_incoming->dev, ifaddr_str, orig_str, prev_sender_str,
         bat_packet->seqno, bat_packet->tq, bat_packet->ttl, bat_packet->
        version, has_directlink_flag);

    hna_buff_len = bat_packet->hna_len * 5;
    hna_recv_buff = (hna_buff_len > 4 ? (unsigned char *)(bat_packet + 1)
        : NULL);

    list_for_each(list_pos, &if_list) {

      batman_if = list_entry(list_pos, struct batman_if, list);

      if (neigh == batman_if->addr.sin_addr.s_addr)
        is_my_addr = 1;

      if (bat_packet->orig == batman_if->addr.sin_addr.s_addr)
        is_my_orig = 1;
        if (neigh == batman_if->broad.sin_addr.s_addr)
          is_broadcast = 1;

        if (bat_packet->prev_sender == batman_if->addr.sin_addr.s_addr)
          is_my_oldorig = 1;

      }


      if (bat_packet->gwflags != 0)
        debug_output(4, "Is an internet gateway (class %i) \n", bat_packet->
            gwflags);

      if (bat_packet->version != COMPAT_VERSION) {
        debug_output(4, "Drop packet: incompatible batman version (%i) \n",
            bat_packet->version);
        goto send_packets;
      }

      if (is_my_addr) {
        debug_output(4, "Drop packet: received my own broadcast (sender: %s)
             \n", neigh_str);
        goto send_packets;
      }

      if (is_broadcast) {
        debug_output(4, "Drop packet: ignoring all packets with broadcast
            source IP (sender: %s) \n", neigh_str);
        goto send_packets;
      }

      if (is_my_orig) {
        orig_neigh_node = get_orig_node(neigh);

        if ((has_directlink_flag) && (if_incoming->addr.sin_addr.s_addr ==
            bat_packet->orig) && (bat_packet->seqno - if_incoming->out.seqno
            + 2 == 0)) {

          debug_output(4, "count own bcast (is_my_orig): old = %i, ",
              orig_neigh_node->bcast_own_sum[if_incoming->if_num]);

          bit_mark((TYPE_OF_WORD *)&(orig_neigh_node->bcast_own[if_incoming
              ->if_num * num_words]), 0);
          orig_neigh_node->bcast_own_sum[if_incoming->if_num] =
              bit_packet_count((TYPE_OF_WORD *)&(orig_neigh_node->bcast_own[
              if_incoming->if_num * num_words]));

          debug_output(4, "new = %i \n", orig_neigh_node->bcast_own_sum[
              if_incoming->if_num]);

        }

        debug_output(4, "Drop packet: originator packet from myself (via
            neighbour) \n");
        goto send_packets;
      }

      if (bat_packet->tq == 0) {
        count_real_packets(bat_packet, neigh, if_incoming);
```

```
    debug_output(4, "Drop packet: originator packet with tq is 0 \n");
    goto send_packets;
  }

  if (is_my_oldorig) {
    debug_output(4, "Drop packet: ignoring all rebroadcast echos (sender
        : %s) \n", neigh_str);
    goto send_packets;
  }


  is_duplicate = count_real_packets(bat_packet, neigh, if_incoming);

  orig_node = get_orig_node(bat_packet->orig);

  /* if sender is a direct neighbor the sender ip equals originator ip
      */
  orig_neigh_node = (bat_packet->orig == neigh ? orig_node :
      get_orig_node(neigh));

  /* drop packet if sender is not a direct neighbor and if we no route
      towards it */
  if ((bat_packet->orig != neigh) && (orig_neigh_node->router == NULL))
      {
    debug_output(4, "Drop packet: OGM via unknown neighbor! \n");
    goto send_packets;
  }

  is_bidirectional = isBidirectionalNeigh(orig_node, orig_neigh_node,
      bat_packet, curr_time, if_incoming);

  /* update ranking if it is not a duplicate or has the same seqno and
      similar ttl as the non-duplicate */
  if ((is_bidirectional) && ((!is_duplicate) ||
        ((orig_node->last_real_seqno == bat_packet->seqno) &&
        (orig_node->last_ttl - 3 <= bat_packet->ttl))))
    update_orig(orig_node, bat_packet, neigh, if_incoming, hna_recv_buff
        , hna_buff_len, is_duplicate, curr_time);



  //ENOTE: Here is our authentication algorithm

  rcvd_challenge = bat_packet->challenge;
  rcvd_response = bat_packet->response;
  rcvd_auth_token = bat_packet->auth_token;

  if(role == 0) { //Unauthenticated node

    if (rcvd_auth_token > 0) {

      if(rcvd_challenge == 0) {

        if(rcvd_response > 0) { //Receive RESPONSE

          tmp_response = (2*generated_request) % UINT8_MAX;
          tmp_response = (tmp_response == 0 ? 1 : tmp_response);
      debug_output(4, "
          ==================================================\n");
      debug_output(4, "[RECV] %d | %d | %d (RESPONSE)\n",
          rcvd_challenge, rcvd_response, rcvd_auth_token);

      if(rcvd_response == tmp_response) { //RESPONSE is correct

        my_challenge = 0;
        my_response = 0;
        my_auth_token = rcvd_auth_token;
        generated_challenge = 0;
        generated_request = 0;
        generated_auth = 0;
        tmp_response = 0;
        role = 1;
        debug_output(4, "[SEND] %d | %d | %d (AUTH)\n", my_challenge
            , my_response, my_auth_token);
        debug_output(4, "YOU ARE AUTHENTICATED!\n");

      } else { //RESPONSE is wrong

        my_challenge = 0;
        my_response = 0;
        my_auth_token = 0;
        tmp_response = 0;
        debug_output(4, "RESPONSE IS WRONG\n");
        tmp_wait = rand() % 10000;
        random_wait_time = curr_time + tmp_wait;

      }
      debug_output(4, "
          ==================================================\n");

    } else { //Receive AUTH

      my_challenge = 0;
      my_response = 0;

      debug_output(4, "
          ==================================================\n");
      debug_output(4, "[RECV] %d | %d | %d (AUTH)\n", rcvd_challenge
          , rcvd_response, rcvd_auth_token);

      if(rcvd_auth_token == generated_auth) { //Receive AUTH (Last
          Message in Handshake)

        role = 2;
        my_auth_token = generated_auth;
        generated_challenge = 0;
        generated_request = 0;
        generated_auth = 0;
        debug_output(4, "YOU ARE MASTER NODE!\n");

      }
      debug_output(4, "
          ==================================================\n");

    }
```

```
          } else {

            if(rcvd_response == 0) { //Receive CHALLENGE FROM MASTER

              if(generated_request == 0) {
                generated_request = 1 + (rand() % UINT8_MAX);
              }
              my_challenge = generated_request;
              my_response = (2*rcvd_challenge) % UINT8_MAX;
              my_response = (my_response == 0 ? 1 : my_response);
              my_auth_token = 0;
              debug_output(4, "
                  ==================================================\n");
              debug_output(4, "[SEND] %d | %d | %d (REQUEST)\n",
                  my_challenge, my_response, my_auth_token);
              debug_output(4, "
                  ==================================================\n");

            }



          }


        } else {
          if(rcvd_challenge == 0) {

            if(rcvd_response == 0) { //Receive PLAIN


              if(curr_time > random_wait_time) {

                debug_output(4, "
                    ==================================================\n");
                debug_output(4, "[RECV] %d | %d | %d (PLAIN)\n",
                    rcvd_challenge, rcvd_response, rcvd_auth_token);

                usleep(rand() % 100000);

                if(generated_challenge==0) {
                  generated_challenge = 1 + (rand() % UINT8_MAX);
                }

                my_challenge = generated_challenge;
                my_response = 0;
                my_auth_token = 0;

                debug_output(4, "[SEND] %d | %d | %d (CHALLENGE)\n",
                    my_challenge, my_response, my_auth_token);
                debug_output(4, "
                    ==================================================\n");

              }

            }

          } else {
```

```
      if(rcvd_response == 0) { //Receive CHALLENGE

        debug_output(4, "
            ==================================================\n");
        debug_output(4, "[RECV] %d | %d | %d (CHALLENGE)\n",
            rcvd_challenge, rcvd_response, rcvd_auth_token);

        if(my_challenge > 0) { //Received CHALLENGE when I have sent
            CHALLENGE (COLLISION)

          my_challenge = 0;
          my_response = 0;
          my_auth_token = 0;

          debug_output(4, "COLLISION!\n");

          tmp_wait = rand() % 10000;
          random_wait_time = curr_time + tmp_wait;

        } else { //Received CHALLENGE

          if((generated_challenge == 0) || (curr_time >
              random_wait_time-(tmp_wait/2))) {
            //if generated_challenge = 0 -> No previously made
                challenges so good to go
            //if more than half of the wait time has passed you start
                accepting challenges

            if(generated_request == 0) {
              generated_request = 1 + (rand() % UINT8_MAX);
            }

            my_challenge = generated_request;
            my_response = (2*rcvd_challenge) % UINT8_MAX;
            my_response = (my_response == 0 ? 1 : my_response);
            my_auth_token = 0;
            debug_output(4, "[SEND] %d | %d | %d (REQUEST)\n",
                my_challenge, my_response, my_auth_token);

          } else {
            debug_output(4, "WAITING\n");
          }

        }
        debug_output(4, "
            ==================================================\n");

      } else { //Receive REQUEST
        tmp_response = (2*generated_challenge) % UINT8_MAX;
        tmp_response = (tmp_response == 0 ? 1 : tmp_response);
        debug_output(4, "
            ==================================================\n");
        debug_output(4, "[RECV] %d | %d | %d (REQUEST)\n",
            rcvd_challenge, rcvd_response, rcvd_auth_token);

        if(rcvd_response == tmp_response) { //REQUEST is correct

          my_challenge = 0;
          my_response = (2*rcvd_challenge) % UINT8_MAX;
```

```
            my_response = (my_response == 0 ? 1 : my_response);

            if(generated_auth == 0) {
              generated_auth = 1 + (rand() % UINT8_MAX);
            }

            my_auth_token = generated_auth;
            debug_output(4, "[SEND] %d | %d | %d (RESPONSE)\n",
                my_challenge, my_response, my_auth_token);

          } else { //REQUEST is wrong

            my_challenge = 0;
            my_response = 0;
            my_auth_token = 0;
            tmp_response = 0;
            debug_output(4, "REQUEST IS WRONG\n");
            tmp_wait = rand() % 10000;
            random_wait_time = curr_time + tmp_wait;

          }
          debug_output(4, "
              ==================================================\n");
        }

      }
    }

    goto send_packets;

  } else if(role == 1) {
    //Authenticated node
    if(rcvd_auth_token != my_auth_token) {
      goto send_packets;
    }


  } else {
    //Master node
    if(rcvd_auth_token == 0) {

      if(rcvd_challenge == 0) {

        if(rcvd_response == 0) { //Receive PLAIN

          debug_output(4, "
              ==================================================\n");
          debug_output(4, "[RECV] %d | %d | %d (PLAIN)\n",
              rcvd_challenge, rcvd_response, rcvd_auth_token);

          if(generated_challenge==0) {
            generated_challenge = 1 + (rand() % UINT8_MAX);
          }

          my_challenge = generated_challenge;
          my_response = 0;

          debug_output(4, "[SEND] %d | %d | %d (CHALLENGE)\n",
              my_challenge, my_response, my_auth_token);
            debug_output(4, "
                ==================================================\n");

          }

        } else {

          if(rcvd_response > 0) { //Received REQUEST

            tmp_response = (2*generated_challenge) % UINT8_MAX;
            tmp_response = (tmp_response == 0 ? 1 : tmp_response);
            debug_output(4, "
                ==================================================\n");
            debug_output(4, "[RECV] %d | %d | %d (REQUEST)\n",
                rcvd_challenge, rcvd_response, rcvd_auth_token);

            if(rcvd_response == tmp_response) { //REQUEST is correct

              my_challenge = 0;
              my_response = (2*rcvd_challenge) % UINT8_MAX;
              my_response = (my_response == 0 ? 1 : my_response);

              debug_output(4, "[SEND] %d | %d | %d (RESPONSE)\n",
                  my_challenge, my_response, my_auth_token);

            }
            debug_output(4, "
                ==================================================\n");

          }

        }

        goto send_packets;

      } else if(rcvd_auth_token != my_auth_token) {
        //Receieve OGM from node in another MANET, auth token > 0, but not
             the same as the masters auth token
        goto send_packets;
      }
    }


    /* is single hop (direct) neighbour */
    if (bat_packet->orig == neigh) {

      /* mark direct link on incoming interface */
      schedule_forward_packet(orig_node, bat_packet, neigh, 1,
          hna_buff_len, if_incoming, curr_time);

      debug_output(4, "Forward packet: rebroadcast neighbour packet with
          direct link flag \n");
      goto send_packets;
    }

    /* multihop originator */
    if (!is_bidirectional) {
      debug_output(4, "Drop packet: not received via bidirectional link\n"
          );
        goto send_packets;
      }

      if (is_duplicate) {
        debug_output(4, "Drop packet: duplicate packet received\n");
        goto send_packets;
      }


      debug_output(4, "Forward packet: rebroadcast originator packet \n");

      schedule_forward_packet(orig_node, bat_packet, neigh, 0, hna_buff_len,
          if_incoming, curr_time);
    }


send_packets:
    send_outstanding_packets(curr_time);

    if ((int)(curr_time - (debug_timeout + 1000)) > 0) {

      debug_timeout = curr_time;

      purge_orig( curr_time );

      debug_orig();

      check_inactive_interfaces();

      if ( debug_clients.clients_num[4] > 0 ) {

        checkIntegrity();
        prof_print();

      }

      if ( ( routing_class != 0 ) && ( curr_gateway == NULL ) )
        choose_gw();

      if ((vis_if.sock) && ((int)(curr_time - (vis_timeout + 10000)) > 0)) {

        vis_timeout = curr_time;
        send_vis_packet();

      }

      hna_local_task_exec();
    }

  }


  if (debug_level > 0)
    printf("Deleting all BATMAN routes\n");

  purge_orig(get_time_msec() + (5 * purge_timeout) + originator_interval);

  hash_destroy(orig_hash);
  list_for_each_safe(list_pos, forw_pos_tmp, &forw_list) {

    forw_node = list_entry(list_pos, struct forw_node, list);

    list_del((struct list_head *)&forw_list, list_pos, &forw_list);

    debugFree(forw_node->pack_buff, 1105);
    debugFree(forw_node, 1106);
  }

  if (vis_packet != NULL)
    debugFree(vis_packet, 1108);

  set_forwarding( forward_old );

  set_rp_filter( if_rp_filter_all_old, "all" );
  set_rp_filter( if_rp_filter_default_old, "default" );

  set_send_redirects( if_send_redirects_all_old, "all" );
  set_send_redirects( if_send_redirects_default_old, "default" );

  return 0;
}
```

C.2 SCHEDULE

C.2.1 schedule.c - excerpt

Line numbers indicate where the code is added to the original source code.

```
75: //ENOTE: Here we add challenge response authentication values to the OGM
     if they are set, node
76: ((struct bat_packet *)forw_node_new->pack_buff)->challenge =
    my_challenge;
78: ((struct bat_packet *)forw_node_new->pack_buff)->response = my_response;
79: ((struct bat_packet *)forw_node_new->pack_buff)->auth_token =
    my_auth_token;
```

C.3 AM

C.3.1 am.h

```
#ifndef AM_H
#define AM_H

#include "batman.h"

extern uint8_t role; // Unauthenticated = 0, Authenticated = 1, Master = 2

extern uint8_t my_challenge; // 0 if no challenge to send
extern uint8_t my_response; // 0 if no response to send
extern uint8_t my_auth_token; // 0 if not authenticated
extern uint8_t tmp_response; // used for response calc.
extern uint8_t generated_challenge; // Verify received response in Request
extern uint8_t generated_request; // Verify received Response
extern uint8_t generated_auth;

extern uint8_t rcvd_challenge; // Received Challenge, 0 if no challenge
extern uint8_t rcvd_response; // Received Response Value, 0 if no response
extern uint8_t rcvd_auth_token; // 0 if not authenticated
extern uint8_t expecting_token; // Expected Value of received auth token

extern uint32_t random_wait_time; // tmp_wait + curr_time
extern uint32_t tmp_wait; // Random backoff time value

#endif
```

C.3.2 am.c

```
#include "am.h"

#include <errno.h>
#include <stdlib.h>
#include <arpa/inet.h>

uint8_t bool_extended = 0;
uint8_t is_authenticated = 0;

uint8_t role = 0;

uint8_t my_auth_token = 0;
uint8_t my_challenge = 0;
uint8_t my_response = 0;
uint8_t rcvd_challenge = 0;
uint8_t rcvd_response = 0;
uint8_t rcvd_auth_token = 0;

uint8_t expecting_token = 0;

uint8_t num_waits = 0;
uint32_t random_wait_time = 0;

uint8_t generated_challenge = 0;
uint8_t generated_request = 0;
uint8_t generated_auth = 0;
uint8_t tmp_response = 0;
uint32_t tmp_wait = 0;
```

Appendix D

Test Results

D.1 Excerpts From Logs

Below are excerpts from the first run of every test.

D.1.1 Test I

This excerpt shows the test runs of test I. The logs are taken from the debug of the master node, which was node 2. The important thing to notice is the timestamp from the first plain OGM received and the timestamp for when it was added to the routing table.

```
[ 1770] Received BATMAN packet via NB: 10.0.0.1, IF: wlan0 10.0.0.2 (from OG
    : 10.0.0.1, via old OG: 10.0.0.1, seqno 1, tq 255, TTL 50, V 9, IDF 0)
[ 1770] Creating new originator: 10.0.0.1
[ 1770] updating last_seqno: old 0, new 1
[ 1770] Creating new last-hop neighbour of originator
[ 1770] bidirectional: orig = 10.0.0.1         neigh = 10.0.0.1         =>
    own_bcast =  0, real recv =  0, local tq:   0, asym_penalty:   0, total
    tq:   0
[ 1770] ==========================================================
[ 1770] [RECV] 0 | 0 | 0 (PLAIN)
[ 1770] [SEND] 86 | 0 | 0 (CHALLENGE)
[ 1770] ==========================================================

    ...
    ...
    ...

[15070] Received BATMAN packet via NB: 10.0.0.1, IF: wlan0 10.0.0.2 (from OG
    : 10.0.0.1, via old OG: 10.0.0.1, seqno 14, tq 255, TTL 50, V 9, IDF 0)
[15070] updating last_seqno: old 13, new 14
[15070] bidirectional: orig = 10.0.0.1         neigh = 10.0.0.1         =>
    own_bcast =  1, real recv = 13, local tq:  19, asym_penalty: 126, total
    tq:   9
[15070] update_originator(): Searching and updating originator entry of
    received packet,
[15070] Updating existing last-hop neighbour of originator
[15070] update_routes()
[15070] Route to 10.0.0.1 via 10.0.0.1
[15070] Adding new route
[15070] Adding route to 10.0.0.1 via 0.0.0.0 (table 66 - wlan0)
```

D.1.2 Test II

This excerpt shows the runs of test II with node 2 as the master node. Note when when the first plain OGM is received and when the new node is added to the routing table.

```
[30460] Received BATMAN packet via NB: 10.0.0.3, IF: wlan0 10.0.0.2 (from OG
    : 10.0.0.3, via old OG: 10.0.0.3, seqno 1, tq 255, TTL 50, V 9, IDF 0)
[30460] Creating new originator: 10.0.0.3
[30460] updating last_seqno: old 0, new 1
[30460] Creating new last-hop neighbour of originator
[30460] bidirectional: orig = 10.0.0.3         neigh = 10.0.0.3         =>
    own_bcast =  0, real recv =  0, local tq:   0, asym_penalty:   0, total
    tq:   0
[30460] ==============================================================
[30460] [RECV] 0 | 0 | 0 (PLAIN)
[30460] [SEND] 232 | 0 | 123 (CHALLENGE)
[30460] ==============================================================

    ...
    ...
    ...

[36280] Received BATMAN packet via NB: 10.0.0.3, IF: wlan0 10.0.0.2 (from OG
    : 10.0.0.3, via old OG: 10.0.0.3, seqno 7, tq 255, TTL 50, V 9, IDF 0)
[36280] updating last_seqno: old 6, new 7
[36280] bidirectional: orig = 10.0.0.3         neigh = 10.0.0.3         =>
    own_bcast =  1, real recv =  6, local tq:  42, asym_penalty:  66, total
    tq:  10
[36280] update_originator(): Searching and updating originator entry of
    received packet,
[36280] Updating existing last-hop neighbour of originator
[36280] update_routes()
[36280] Route to 10.0.0.3 via 10.0.0.3
[36280] Adding new route
[36280] Adding route to 10.0.0.3 via 0.0.0.0 (table 66 - wlan0)
```

D.1.3 Test III

Here is the results from test III for both our implementation and the original BATMAN protocol. Here we show that the last OGM received from node 4 before it was removed. Notice the arrival timestamp of this OGM and of the route change.

Our Implementation

```
[906250] Received BATMAN packet via NB: 10.0.0.4, IF: wlan0 10.0.0.1 (from
    OG: 10.0.0.1, via old OG: 10.0.0.1, seqno 897, tq 240, TTL 49, V 9, IDF
    1)
[906250] count own bcast (is_my_orig): old = 62, [    906250] new = 63
[906250] Drop packet: originator packet from myself (via neighbour)
[906250]
[906300] Received BATMAN packet via NB: 10.0.0.2, IF: wlan0 10.0.0.1 (from
    OG: 10.0.0.2, via old OG: 10.0.0.2, seqno 960, tq 255, TTL 50, V 9, IDF
    0)
[906300] updating last_seqno: old 959, new 960
```

```
[906300] bidirectional: orig = 10.0.0.2        neigh = 10.0.0.2        =>
    own_bcast = 35, real recv = 41, local tq: 217, asym_penalty: 244, total
    tq: 207
[906300] update_originator(): Searching and updating originator entry of
    received packet,
[906300] Updating existing last-hop neighbour of originator
[906300] update_routes()
[906300] schedule_forward_packet():
[906300] forwarding: tq_orig: 207, tq_avg: 217, tq_forw: 212, ttl_orig: 49,
    ttl_forw: 48
[906300] Forward packet: rebroadcast neighbour packet with direct link flag
[906300] ----------------------- DEBUG -----------------------
[906300] Forward list
[906300]     10.0.0.2 at 906419
[906300]     10.0.0.1 at 907160
[906300] Originator list
[906300]   Originator  (#/255)         Nexthop [outgoingIF]:    Potential
    nexthops
[906300]       10.0.0.4 (249)        10.0.0.4 [     wlan0], last_valid:
    905630:
[906300]        10.0.0.4 (249)        10.0.0.2 (181)
[906300]       10.0.0.2 (217)        10.0.0.4 [     wlan0], last_valid:
    906300:
[906300]        10.0.0.2 (199)        10.0.0.4 (217)
[906300] ------------------------------------------------------------ END DEBUG

    ...
    ...
    ...

[924070] Received BATMAN packet via NB: 10.0.0.2, IF: wlan0 10.0.0.1 (from
    OG: 10.0.0.1, via old OG: 10.0.0.1, seqno 914, tq 142, TTL 49, V 9, IDF
    1)
[924070] count own bcast (is_my_orig): old = 35, [    924070] new = 36
[924070] Drop packet: originator packet from myself (via neighbour)
[924070] ----------------------- DEBUG -----------------------
[924070] Forward list
[924070]     10.0.0.1 at 925084
[924070] Originator list
[924070]   Originator  (#/255)         Nexthop [outgoingIF]:    Potential
    nexthops
[924070]       10.0.0.4 (249)        10.0.0.4 [     wlan0], last_valid:
    905630:
[924070]        10.0.0.4 (249)        10.0.0.2 (181)
[924070]       10.0.0.2 (219)        10.0.0.4 [     wlan0], last_valid:
    923540:
[924070]        10.0.0.2 (214)        10.0.0.4 (219)
[924070] ------------------------------------------------------------ END DEBUG
[924070]
[924510] Received BATMAN packet via NB: 10.0.0.2, IF: wlan0 10.0.0.1 (from
    OG: 10.0.0.2, via old OG: 10.0.0.2, seqno 978, tq 255, TTL 50, V 9, IDF
    0)
[924510] updating last_seqno: old 977, new 978
[924510] bidirectional: orig = 10.0.0.2        neigh = 10.0.0.2        =>
    own_bcast = 36, real recv = 36, local tq: 255, asym_penalty: 234, total
    tq: 234
[924510] update_originator(): Searching and updating originator entry of
    received packet,
[924510] Updating existing last-hop neighbour of originator
```

```
[924510] update_routes()
[924510] Route to 10.0.0.2 via 10.0.0.2
[924510] Route changed
```

Original BATMAN

```
[268760] Received BATMAN packet via NB: 10.0.0.4, IF: wlan0 10.0.0.1 (from
    OG: 10.0.0.1, via old OG: 10.0.0.1, seqno 265, tq 240, TTL 49, V 5, IDF
    1)
[268760] count own bcast (is_my_orig): old = 60, [     268760] new = 61
[268760] Drop packet: originator packet from myself (via neighbour)
[268760] ------------------- DEBUG -------------------
[268760] Forward list
[268760]     10.0.0.1 at 269694
[268760] Originator list
[268760]   Originator  (#/255)          Nexthop [outgoingIF]:    Potential
    nexthops
[268760]        10.0.0.4 (243)        10.0.0.4 [     wlan0], last_valid:
    268060:
[268760]         10.0.0.4 (243)        10.0.0.2 (188)
[268760]        10.0.0.2 (216)        10.0.0.4 [     wlan0], last_valid:
    268280:
[268760]         10.0.0.2 (214)        10.0.0.4 (216)
[268760] ------------------------------------------------- END DEBUG

    ...
    ...
    ...

[288110] Received BATMAN packet via NB: 10.0.0.2, IF: wlan0 10.0.0.1 (from
    OG: 10.0.0.1, via old OG: 10.0.0.1, seqno 284, tq 124, TTL 49, V 5, IDF
    1)
[288110] count own bcast (is_my_orig): old = 33, [     288110] new = 34
[288110] Drop packet: originator packet from myself (via neighbour)
[288110] ------------------- DEBUG -------------------
[288110] Forward list
[288110]     10.0.0.1 at 289060
[288110] Originator list
[288110]   Originator  (#/255)          Nexthop [outgoingIF]:    Potential
    nexthops
[288110]        10.0.0.4 (243)        10.0.0.4 [     wlan0], last_valid:
    268060:
[288110]         10.0.0.4 (243)        10.0.0.2 (188)
[288110]        10.0.0.2 (223)        10.0.0.4 [     wlan0], last_valid:
    286540:
[288110]         10.0.0.2 (215)        10.0.0.4 (223)
[288110] ------------------------------------------------- END DEBUG
[288110]
[289060]
[289070] Sending own packet (originator 10.0.0.1, seqno 285, TQ 255, TTL 50,
     IDF off) on interface wlan0
[289070] schedule_own_packet(): wlan0
[289070] count own bcast (schedule_own_packet): old = 43, [     289070] new =
     43
[289070] count own bcast (schedule_own_packet): old = 34, [     289070] new =
     33
[289070]
```

```
[289070] Received BATMAN packet via NB: 10.0.0.1, IF: wlan0 10.0.0.1 (from
   OG: 10.0.0.1, via old OG: 10.0.0.1, seqno 285, tq 255, TTL 50, V 5, IDF
   0)
[289070] Drop packet: received my own broadcast (sender: 10.0.0.1)
[289070]
[289670] Received BATMAN packet via NB: 10.0.0.2, IF: wlan0 10.0.0.1 (from
   OG: 10.0.0.2, via old OG: 10.0.0.2, seqno 1391, tq 255, TTL 50, V 5, IDF
   0)
[289670] updating last_seqno: old 1388, new 1391
[289670] bidirectional: orig = 10.0.0.2        neigh = 10.0.0.2        =>
   own_bcast = 33, real recv = 34, local tq: 247, asym_penalty: 229, total
   tq: 221
[289670] update_originator(): Searching and updating originator entry of
   received packet,
[289670] Updating existing last-hop neighbour of originator
[289670] update_routes()
[289670] Route to 10.0.0.2 via 10.0.0.2
[289670] Route changed
```

D.1.4 Test IV

The excerpts from test IV shows shows how long time it takes from the first OGM from node 4 is received, until it is added as the preferred route again.

Our Implementation

```
[1306170] Received BATMAN packet via NB: 10.0.0.4, IF: wlan0 10.0.0.1 (from
   OG: 10.0.0.1, via old OG: 10.0.0.1, seqno 1292, tq 0, TTL 49, V 9, IDF 1)
[1306170] Creating new originator: 10.0.0.4
[1306170] count own bcast (is_my_orig): old = 0, [   1306170] new = 1
[1306170] Drop packet: originator packet from myself (via neighbour)
[1306170]
[1306770] Received BATMAN packet via NB: 10.0.0.4, IF: wlan0 10.0.0.1 (from
   OG: 10.0.0.2, via old OG: 10.0.0.2, seqno 1355, tq 0, TTL 49, V 9, IDF 1)
[1306770] updating last_seqno: old 1354, new 1355
[1306770] Drop packet: originator packet with tq is 0
[1306770] Originator timeout: originator 10.0.0.4, last_valid 0
[1306770] update_routes()
[1306770] --------------- DEBUG ---------------
[1306770] Forward list
[1306770]     10.0.0.1 at 1307170
[1306770] Originator list
[1306770]   Originator  (#/255)        Nexthop [outgoingIF]:   Potential
   nexthops
[1306770]        10.0.0.2 (226)        10.0.0.2 [     wlan0], last_valid:
   1305720:
[1306770]         10.0.0.2 (226)
[1306770] ---------------------------------------- END DEBUG

     ...
     ...
     ...

[1347570] Received BATMAN packet via NB: 10.0.0.4, IF: wlan0 10.0.0.1 (from
   OG: 10.0.0.2, via old OG: 10.0.0.2, seqno 1395, tq 220, TTL 49, V 9, IDF
   1)
```

```
[1347570] bidirectional: orig = 10.0.0.2        neigh = 10.0.0.4        =>
    own_bcast = 39, real recv = 39, local tq: 255, asym_penalty: 240, total
    tq: 207
[1347570] update_originator(): Searching and updating originator entry of
    received packet,
[1347570] Updating existing last-hop neighbour of originator
[1347570] update_routes()
[1347570] Route to 10.0.0.2 via 10.0.0.4
[1347570] Route changed
```

Original BATMAN

```
[471880] Received BATMAN packet via NB: 10.0.0.4, IF: wlan0 10.0.0.1 (from
    OG: 10.0.0.1, via old OG: 10.0.0.1, seqno 466, tq 0, TTL 49, V 5, IDF 1)
[471880] count own bcast (is_my_orig): old = 0, [    471880] new = 1
[471880] Drop packet: originator packet from myself (via neighbour)
[471880] ---------------- DEBUG ----------------
[471880] Forward list
[471880]     10.0.0.1 at 472727
[471880] Originator list
[471880]   Originator  (#/255)         Nexthop [outgoingIF]:   Potential
    nexthops
[471880]        10.0.0.2 (167)         10.0.0.2 [     wlan0], last_valid:
    469460:
[471880]         10.0.0.2 (167)
[471880] ---------------------------------------- END DEBUG


    ...
    ...
    ...

[501730] Received BATMAN packet via NB: 10.0.0.4, IF: wlan0 10.0.0.1 (from
    OG: 10.0.0.2, via old OG: 10.0.0.2, seqno 1601, tq 181, TTL 49, V 5, IDF
    1)
[501730] bidirectional: orig = 10.0.0.2        neigh = 10.0.0.4        =>
    own_bcast = 28, real recv = 28, local tq: 255, asym_penalty: 210, total
    tq: 149
[501730] update_originator(): Searching and updating originator entry of
    received packet,
[501730] Updating existing last-hop neighbour of originator
[501730] update_routes()
[501730] Route to 10.0.0.2 via 10.0.0.4
[501730] Route changed
```

D.2 Full Logs

The full logs of all the test runs can be downloaded from http://org.ntnu.no/batman2010/test_logs.tar.gz. Here you will find every test done with our implementation in the root folder, and the tests done on the original BATMAN protocol inside the "original_batman" folder. Every file is named with the test number, run number and the node the debug log is taken from as follows - test_run_node.txt. E.g. the file called "2_2_1.txt" contains the log of run number two of test two taken from node one.

Appendix E

TTM4137 Essay

This essay served as my (Espen Graarud) technical essay for the course TTM4137 Wireless Network Security at NTNU. The essay lay some of the groundwork for this project with the title: ”The Authentication Process in Mobile Ad Hoc Networks”.

The essay starts at the following page.

The Authentication Process in Mobile Ad Hoc Networks

Espen Grannes Graarud - espengra@stud.ntnu.no

TTM4137 Wireless Security Technical Essay - November 15, 2010

Introduction

A Mobile Ad Hoc Network (MANET) is a network without an infrastructure where the participating nodes may come and go rapidly and in an unpredictable fashion [1]. To maintain the network an ad hoc routing protocol must be present, but most protocols do not provide sufficient security. This is a huge concern in such networks because of the possibility of e.g. route and node spoofing [2]. To eliminate such threats a good authentication scheme should be in place, and I will propose a general solution for such a scheme in this essay.

Problem Discussion

Spoofing An adversary can perform route and node spoofing quite easliy if there is no good authentication scheme in place[3]. We need an authentication system to make sure nodes can properly verify their identity so that you can trust the intermediate nodes between you and the end-node you want to communicate with, and the end-node itself. If you cannot trust all the nodes in the network, you cannot be sure your packet is ever sent to the intended recipient, altered on its way, or copied and sent to someone else as well [4].

PKI If you authenticate all nodes in the network, or at a minimum the nodes between you and the end-node that you wish to communicate with and the end-node itself you should be sufficiently safe. On the Internet, the most common way of authenticating nodes or users is to use public key certificates and a public key infrastructure (PKI) [5].

When you know the certificate of all required nodes (see above), you can verify their identity with a challenge-response. As long as you can trust the scheme of signing certificates and broadcasting the certificates to all nodes you can trust the challenge-response of the nodes in the network. On the Internet you typically trust a few Certificate Authorities (CA) by default[6] and therefore trust certificates signed by them, or by other CAs that are signed by CAs you already trust.

Problem with Ad Hoc Networks So why not just incorporate a PKI in the MANET? If you have a stationary ad hoc network this would be a plausible solution, but for MANETs, you have situations where you cannot reach the CA. Because of the diffucult nature of mobile networks, the CA might lose its connection to the network from time to time. E.g. it can move out of range or just get a huge amount of packet collisions[1]. The CA might also be the point of attack if an attacker is satisfied with a denial of service attack (DoS). If you then rely on a single CA to authenticate new nodes' certificates but the CA is offline, you simply cannot verify the new nodes' identity and you cannot allow them access to the network. This of course, can be a huge problem in MANETs.

Solution Proposal

Here I propose a simplified general solution where I combine properties of PKI and Web of Trust (WOT) [7] like this:

I A CA signs new certificates.

II The CA periodically broadcasts a list of known nodes to all the nodes in the MANET.

III All nodes save the list of known nodes locally.

IV If the CA goes offline, one known node assumes the role to broadcast the list of known nodes to the MANET.

With this combination nodes only trust the CA to sign new certificates, but they trust their already known nodes to verify other nodes known to the MANET when the CA is down. This is an important feature because already known nodes that are offline should be able to communicate with nodes that has been verified while they were offline, even if the CA is down.

A simple example to show this feature is shown in Figure 1. Here we see that Node A knows the CA and Node B when it disconnects. While Node A is offline a new Node C is verified by the CA, and the CA goes offline before Node A connects to the MANET again. Now, because Node B has assumed the role of broadcasting the known nodes list, Node A learns to know Node C and they can communicate from this point on - without the help of the CA.

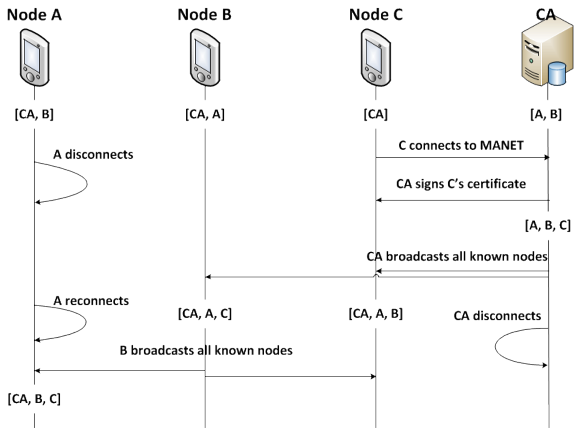


Figure 1: Node A learn the new node Cs identity through a broadcast by node B because the CA is down while node A reconnects. The brackets show who each node knows

Because the CA broadcasts the list of known nodes periodically, all nodes in the MANET can assume that the CA is offline if they do not receive a list after some predefined time. The timeout selection should be a function of the time-interval between the periodic broadcasts sent by the CA. Also, it should be noted that a low timeout threshold does not imply too much damage to the performance of the network as the new broadcasts sent by one of the nodes does not change the state of the network, it only maintains the state and when the CA returns it can assume operation as normal. To have a too high threshold would potentially do more damage as many nodes might reconnect to the network within the threshold time and therefore not receive the new known nodes list.

The question of which node should assume the role to broadcast the list is not trivial. From this point on I refer to this node as a ”master node”. We can assume that all nodes online at the moment the CA went offline possess the same list of known nodes, the last complete list broadcasted by the CA. One might be led to believe that all of these nodes are potentially good master nodes, but the question isn’t really how many nodes they know and trust, rather than how many nodes know and trust them.

If there has been many nodes dropping out within a short time frame before a node gets authenticated by the CA, they will not know about this node, and if they reconnect after the CA disconnects, they will not know or trust this node. Therefore they will disregard this nodes broadcasts if it should assume the role as a master node. Because of this property, the selection of a master node must not only consider how many nodes the future master node knows, but also how many nodes know the master node itself.

Limitations

My intentions has been to propose a general solution, and therefore I cannot assume too much of the usage. Here are some of the limitations my proposal does not cover.

Untrusted Master Node If a node does not know the current master node, he will not trust its broadcasted known nodes list. We could take advantage of more WOT properties and decide that if many known nodes of one node trust the master node, he too will trust the master node.

No New Nodes As long as the CA is down no new nodes will be trusted to communicate with the network. Here too, WOT properties could be used so that the master node could potentially sign new nodes certificates. However it would be very difficult to establish the real identity of the node without an out-of-band authentication.

Initial Trust My solution needs some kind of initial trust for the CA to accept new nodes. The node either needs some long lived public key certificate signed by another trusted CA, or we need some out-of-band authentication like a smart card for instance.

Conclusion

In this essay I have proposed a general solution for authentication in MANETs. The general idea in my solution can be used with most current protocols and can be adjusted for for different kinds of scenarios. Mobile ad hoc networks are becoming more and more used, especially in military and emergency situations, and the need for a good authentication scheme is therefore most wanted. I have also shown that the authentication process for MANETs is far more complex than on regular networks with consistent infrastructure because of the flat topology of the MANET.

Acknowledgments

I would like to acknowledge significant contributions to the ideas in this paper by Anne Gabrielle Bowitz, my partner in a related project where we add authentication mechanisms to the B.A.T.M.A.N. protocol.

References

[1] J. Sheu and W. Jie, *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, p. 178. Auerbach Publishers, 2005.

[2] J.-P. Hubaux, L. Buttyán, and S. Capkun, “The quest for security in mobile ad hoc networks,” in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc ’01, (New York, NY, USA), pp. 146–155, ACM, 2001.

[3] K. Sanzgiri, B. Dahill, B. Levine, C. Shields, and E. Belding-Royer, “A secure routing protocol for ad hoc networks,” 2002.

[4] B. Wu, J. Chen, J. Wu, and M. Cardei, “A survey of attacks and countermeasures in mobile ad hoc networks,” *Wireless Network Security*, pp. 103–135, 2007.

[5] R. Housley, W. Polk, W. Ford, and D. Solo, “Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile,” 2002.

[6] K. Wilson, “Included certificate list,” *Mozilla*, June 2009. Last visited: 12 Nov. 2010. http://www.mozilla.org/projects/security/certs/included/.

[7] G. Caronni, “Walking the web of trust,” in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000. (WET ICE 2000). Proeedings. IEEE 9th International Workshops on*, 2000.